

ASCII SYSTEMSOFT

PCファミリー・テクニカル・ノウハウ集 PC-6000シリーズ編第1巻

PC-Techknow^{テク}6000^{ノウ}Vol.1

一零・樋口理・八木良一共著 システムソフト監修



アス+

アスキー・システムソフトシリーズ
PCファミリー・テクニカル・ノウハウ集

PC-Techknow6000Vol.I

共著／一 零 樋口 理

八木良一

監修／システムソフト



アスキー出版



システムソフト

アスキー・システムソフトシリーズ

PC ファミリー・テクニカル・ノウハウ集の発刊にあたって

株式会社システムソフト福岡では、株式会社アスキー出版と共同で、「アスキー・システムソフトシリーズ、PCファミリー・テクニカル・ノウハウ集」を発刊することになりました。

この、「PC-Techknow シリーズ」は、NEC のパーソナルコンピュータ、PC ファミリー (PC-8001, PC-6001, PC-8801) および、その周辺機器を徹底的に活用するためのテクニカル・ノウハウ (Techknow テクノウ) をまとめたもので、構成は以下のようになっています。

- PC-8000シリーズ編 (N-BASIC)
- PC-6000シリーズ編 (N₆₀-BASIC)
- PC-8800シリーズ編 (N₈₈-BASIC)

各シリーズ共、2巻程度にまとめる予定です。

内容は、それぞれの BASIC の内部構造から、キー入力の仕方、カセットおよびディスクファイルの上手な使用法、BASIC プログラム・テクニック、さらには、機械語理解のポイントまで、活用いただける情報が満載されています。

本書は、PC-6000 シリーズ第1巻の「PC-Techknow6000 Vol.1」であり、2巻以後も順次発刊の予定です。

プログラム入力時の注意.

本書に掲載されているBASICプログラム中、特に断りのないものは、MP-80を使用して出力したもので、CRT画面や、他のプリンターで打ち出したものとは出力形式が若干異なっています。

特に、1行が2段にわたるプログラムを入力する際、行番号の下空白はつめて打ち込んで下さい。

はじめに

半導体技術の急速な進歩は、低価格のパーソナルコンピュータを生み出し、私たちの日常生活にまでかかわりをもつようになってきています。現在、各メーカーからいろいろな機種が発売されていますが、その中でも日本電気株式会社の PC-6000、PC-8000、PC-8800 の 3 シリーズはパーソナルコンピュータのひとつの形を定めたといえるでしょう。

この PC ファミリーのうち、ホームコンピュータとして使えるように考えられた PC-6001 は、10万円台を割る低価格にもかかわらず、非常に高い機能が盛り込まれています。しかしその機能を十二分に引き出し、有効に使うかどうかは、ユーザー自身の態度いかんにかかっています。

PC-6001 を調べるにつれて、マニュアルに書かれていない有用な機能が多数あることに気付きました。また、PC-8001 に比べて機能が多少割愛された部分もあります。有用な機能を引き出し、割愛された機能を補うためには、どうしても N₆₀-BASIC の内部を知る必要があります。また、機械語を使用する場合においても内部構造を熟知していなければなりません。

本書は、こうした点に主眼をおいて PC-6001 を徹底的に活用するためのテクニカル・ノウハウをまとめたものです。読者の方々にパーソナルコンピュータに対する理解をより深めていただき、自由自在に使いこなすための書として利用していただければ幸いに存じます。

なお、本書を執筆するに際して、第 5 章を樋口理、N₆₀-BASIC の解析および付録を八木良一、他を一零が担当しました。さらに、システムソフト・スタッフが編集・監修を行ないました。

末筆ながら、著者らが本書の出版にあたり、大変御世話になりました株式会社アスキー出版のスタッフの方々ならびに株式会社システムソフト福岡の樺島社長、藤田出版部長に心から感謝いたします。

1982年 7 月

著者代表

かずと れい
一 零

目 次

PCファミリー・テクニカル・ノウハウ集の発刊にあたって	2
はじめに	3

第1章 PC-6001のハードウェア仕様

11

1-1 本体ブロック図	11
1-2 システム仕様	11
1-3 プログラムエリア	14
1-4 ROM & RAMカートリッジ	15
1-5 I/Oマップ	18

第2章 N60-BASICの内部構造

21

2-1 メモリ・マップ	21
2-2 ユーザーエリア	22
2-3 プログラムの格納状態	24
2-4 プログラムの回復のさせ方	28
2-5 中間言語	31
2-6 中間言語処理ルーチン	35
2-7 識別コード	38
2-8 単変変数領域	39
2-9 配列変数領域	41
2-10 文字変数と文字領域	43
2-11 フリーエリア	46
2-12 浮動小数点表記法	49

第3章 CRTディスプレイ

53

3-1 VDG(ビデオディスプレイジェネータ)	53
3-2 アトリビュート	55
3-3 VRAMのアドレス	59

3-3-1	アトリビュートアドレスマップ	59
3-3-2	テキスト・セミグラフィックアドレスマップ	59
3-3-3	グラフィックアドレスマップ	60
3-4	キャラクタ・ジェネレータ	62
3-5	ページの切り換え	63
3-6	表示期間	65

第4章 キー入力 69

4-1	ファンクションキー	69
4-1-1	メモリの格納状態	69
4-1-2	ROM内の格納状態	69
4-1-3	KEY LIST	70
4-1-4	内容の定義の仕方	70
4-1-5	キーポインタとファンクションキーフラグの使い方	71
4-2	キー入力カステートメント	73
4-2-1	INPUT	73
4-2-2	INKEY\$	74
4-2-3	STICK, STRIG	74
4-2-4	キーバッファ	76
4-3	コントロールキー	77

第5章 サウンド機能 81

5-1	PLAY命令の基礎	81
5-2	SOUND命令	88
5-3	より高度なテクニック	93
5-3-1	PLAYのバッファ	93
5-3-2	PLAYに変数を	94
5-3-3	PLAYと音色	95
5-3-4	サウンド機能と機械語	97

目 次

第6章 カセット —————103

6-1	ボーレート	103
6-2	フォーマット	104
6-2-1	プログラムファイル	104
6-2-2	データ・ファイル	105
6-3	PC-8001のデータをPC6001で使用する	106
6-4	PC-8001のプログラムテープを PC-6001でLOADする	108
6-5	BASICと機械語を一度にSAVE・LOADする	110
6-6	データ・ファイルにおける，と，との違い	111

第7章 プリンタ出力 —————117

7-1	PC-6021	117
7-2	キャラクタ	118
7-3	画面コピーの方法	120
7-3-1	PC-6021による画面コピー	120
7-3-2	他のプリンタによる画面コピー	121
7-3-3	キャラクタのみ画面コピーする方法	122
7-4	PC-8023によるひらがな出力	123
7-5	PC-8023によるグラフィック出力	124

第8章 N₈₀-BASICの命令分析 —————129

8-1	N ₈₀ -BASICとN-BASICの命令比較	129
8-2	いろいろな命令	131
8-2-1	LINE	131
8-2-2	PSET, PRESET	132
8-2-3	COLOR	132

8-2-4	CLS	134
8-3	色のつけ方	136
8-4	N60-BASICにない命令をある命令で代用する	140

第9章 EXECとUSR143

9-1	モニタ	143
9-2	EXECとUSR	144
9-2-1	EXEC	144
9-2-2	EXECの応用	145
9-2-3	USR	147
9-3	引数	148
9-3-1	数値型	148
9-3-2	文字型	149
9-4	BASICを機械語で	150
9-4-1	ファンクションキー・イニシャライズ	150
9-4-2	KEY LIST	151
9-4-3	日本語エラーメッセージ	151

第10章 ランダムテクニック155

10-1	TIME	155
10-1-1	タイム機能	155
10-1-2	タイマのセット	155
10-2	知っていればおもしろいランダムテクニック	158
10-2-1	CLOAD PRINT	158
10-2-2	GOTO	158
10-3	アンリストの方法	159
10-4	SCREENのもう一つの使い方	162
10-5	画面を消して実行速度アップ	163
10-6	1行は71文字以上可能か?	164

目 次

10-7 拡張ROMエリアの使い方	165
10-8 PRINTとLPRINTの切り換え	166
10-9 PEEK, ROKEを使って省メモリ化	167
10-10 アベンド	168
10-11 行番号を0にする方法	170
10-12 PRESETをPSETとしても使える	171
10-13 グラフィックで相対座標が使える	172
10-14 エラーの音を変えてみよう	174
10-15 SOUND, PLAY関係のデフォルト値	175

付 録

177

付-1 I/Oポート一覧表	179
付-2 N60-BASICインプリンター一覧表	181
付-3 ワークエリア一覧表	210
付-4 中間言語と処理ルーチン対応表	226
付-5 キャラクタ・コード表	228
付-6 キーボード配列表	229
付-7 エラーメッセージ一覧表	231
付-8 タイニー・モニタ	232
付-9 12平均率音階表	234
付-10 サウンドレジスター一覧表	235

索 引

236

第1章 PC-6001のハードウェア仕様

- 1-1 本体ブロック図
- 1-2 システム仕様
- 1-3 プログラムエリア
- 1-4 ROM & RAMカートリッジ
- 1-5 I/Oマップ

第1章 PC-6001のハードウェア仕様

1-1 本体ブロック図

本体のブロック図を次のページに示します。

1-2 システム仕様

(1) CPU

μPD780C-1(Z-80 コンパチブル)3.9936MHz

(2) メモリ

○ROM……N₆₀-BASIC インタプリク 16KB(μPD2364×2 マスクNO. 677, 678)

オプション 16KB(ROM カートリッジ)

○RAM……16KB(μPD416-3×8)

オプション 16KB(ROM & RAM カートリッジ)

(3) CRT

○VDG(ビデオディスプレイジェネレータ)……M5C6847P-1(MC6847 コンパチブル)

○キャラクタ・ROM……4KB(μPD2332×1 マスク NO. 414)

○スクリーン構成……32文字×16行 (512文字)

○文字構成……文字+グラフィック記号248種(7×9ドット)

○グラフィック……256×192ドット

128×192ドット

64×48ドット

※グラフィックと文字の混在可能

○カラー……最大9色(黒, 緑, 黄, 青, 赤, 白, シアン, マゼンダ, オレンジ)

○ページ数……最大4ページ

(4) サウンド機能

○AY-3-8910 使用(クロック 3.579545MHz)

○音階……8オクターブ(3重和音出力可能)

○特殊効果音は8910の機能に準ず

(5) カセットインタフェイス

○FSK 方式

○600ボー, 1200ボー切り換え可

○リモート機能有

PC-6001本体

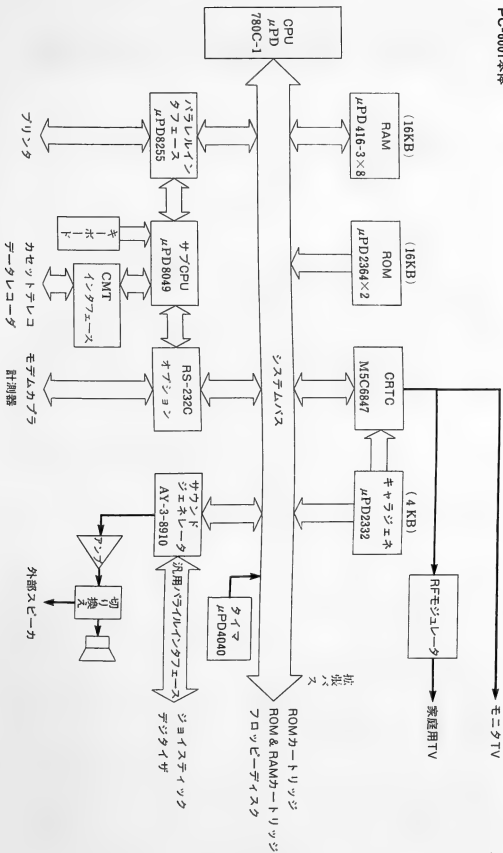


図1-1 PC-6001本体ブロック図

(6) プリンタ・インタフェイス

○TTLレベル8ビットパラレル(セントロニクス社仕様準拠)

○JIS コード準拠

(7) 汎用バス

○TTL レベルパラレルポート

ジョイスティック、デジタイザ等接続可

(8) CRTインタフェイス

○コンポジットビデオ信号出力方式(専用ディスプレイ)

○NTSC 出力方式(家庭用テレビ 1ch, または 2ch 使用)

(9) タイマ機能

○約 2ms ごとのインターバルタイマ(禁止可)

(10) RS-232C インタフェイス(オプション)

○EIA RS-232C 準拠

300, 600, 1200, 2400, 4800ボー選択可

(11) キーボード

○ μ PD8049 によるソフトウェアスキャン

○71キー JIS 標準配列準拠

○コントロールキー, ファンクションキー有

○リピート機能有

(12) 拡張バス

○2.54mm ピッチ 50 ピンバス

1-3 プログラムエリア

(1) 16K バイトの場合

32K バイトの場合

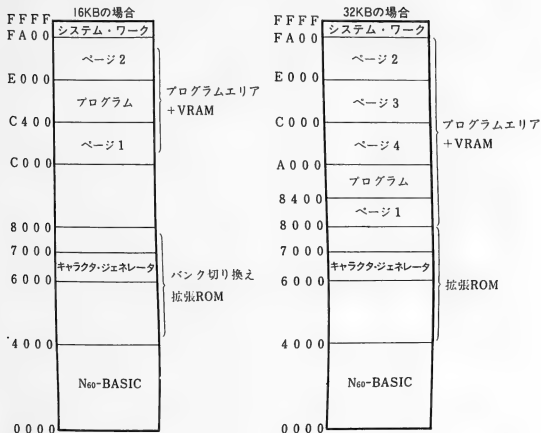


図 1-2 メモリ・マップ

(2) プログラムエリアは使用するページ数で変わります。

ページ数	1	2	3	4
RAM				
16KB	C400 ~ F9FFH	C400 ~ DFFFH		
32KB	8400 ~ F9FFH	8400 ~ DFFFH	8400 ~ BFFFH	8400 ~ 9FFFH

図 1-3 プログラムエリアとページ数

1-4 ROM & RAM カートリッジ

(1) ユーザーエリア

PC-6006(ROM & RAM カートリッジ)を使用することにより RAM を 16K バイト増やすことができます。また、ROM を最大 32K バイトまで実装することが可能です。

RAM ページ数	32KB(増設有)	16KB(増設なし)
1	30140バイト	13756バイト
2	23484 "	7100 "
3	15292 "	
4	7100 "	

図1-4 PC-6006とユーザーエリアとの関係

(2) ROM の使用

ROM はマスク ROM として μ P D2316, μ PD2332, μ PD2364が使用できます。

われわれユーザーが利用する場合は、PROM を使用することになります。PROM として使用できるのは、ROM & RAM カートリッジのマニュアルによれば μ PD2732 となっていますが、 μ PD2716, TMS2532, TMS2564 等も使用可能です。

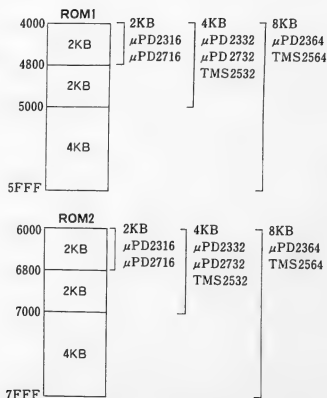
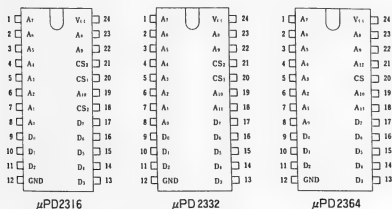


図1-5-1 拡張ROMメモリ・マップ

ROMサイズ	ROM名	スイッチの位置
2 KB	μ PD2316	下
	μ PD2716	下
4 KB	μ PD2332	下
	μ PD2732	上
8 KB	TMS2532	下
	TMS2564	下

図1-5-2 ROMセレクトスイッチの位置



マスクROMのピン配置

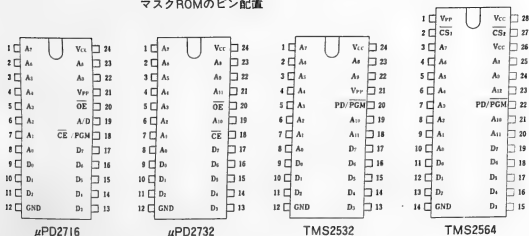


図1-6 ROMのピン配置

(3) ROM を取り付け方向

ROM & RAM カートリッジのマニュアルには ROM を取り付け方向の説明がありません。ROM の方向を間違えますと ROM が破損する原因になります。方向を間違えないように、また、ROM の足を曲げたりしないように注意深く取り付ける必要があります。

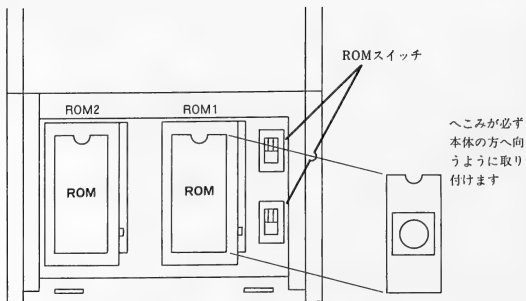


図1-7 ROMの取り付け方向

※PROM の種類によってはジャンパ線等を取り付ける必要があります。

1-5 I/O マップ

FF	未使用	メーカーで使用	$\overline{\text{RAS}}$	1	2	ROM2 $\overline{\text{OE}}$
			$\overline{\text{DBIN}}$	3	4	ROM1 $\overline{\text{OE}}$
			$\overline{\text{WE}}$	5	6	N.C.
D0			N.C.	7	8	N.C.
		プリンタBUSY	N.C.	9	10	MPX
C0			N.C.	11	12	N.C.
		タイマ	N.C.	13	14	N.C.
B0		ページ切り換え、モータ	N.C.	15	16	N.C.
	8910	SOUND	A9	17	18	N.C.
		ジョイスティック	A11	19	20	A10
A0			A7	21	22	A6
	8255	プリンタ、8049	A13	23	24	A8
90		CGROMセレクト	N.C.	25	26	A12
			A1	27	28	A0
	8251	RS-232C	A3	29	30	A2
			A5	31	32	A4
80			D1	33	34	D0
			D3	35	36	D2
			D5	37	38	D4
			D7	39	40	D6
	未使用	ユーザーで使用可	GND	41	42	N.C.
			GND	43	44	RAM-ROM CARD ON
			+5V	45	46	RAM-ROM CARD ON
			+5V	47	48	+12V
			-5V	49	50	N.C.

本体挿入口
から見て上面

図1-8 I/Oマップ

図1-8-1 拡張バス信号

80~CFH までを PC-6001 の内部で使用しています。D0~FFH までは、ディスク等の周辺機器に使用すると思われます。00~7FH まではユーザー開放されたエリアで、それを使用するための外部バスの信号は以上になっています。

※I/O の機能については各章で説明が加えられています。

第2章 N₈₀-BASICの内部構造

- 2-1 メモリ・マップ
- 2-2 ユーザーエリア
- 2-3 プログラムの格納状態
- 2-4 プログラムの回復のさせ方
- 2-5 中間言語
- 2-6 中間言語処理ルーチン
- 2-7 識別コード
- 2-8 単変数領域
- 2-9 配列変数領域
- 2-10 文字変数と文字列領域
- 2-11 フリーエリア
- 2-12 浮動小数点表記法

第2章 N₆₀-BASIC の内部構造

2-1 メモリ・マップ

PC-6001 内部のメモリ空間は、64K バイトあり、前半の 32K バイトが ROM、後半 32K バイトが RAM に割り当てられており、6000~6FFFH をバンク切り換えでキャラジェネのエリアとして使用しています。メモリ・マップは RAM の拡張やスクリーンのページ数によって変わりますが、とくにことわりがない場合を除いて RAM32K バイト、ページ 2 で説明します。

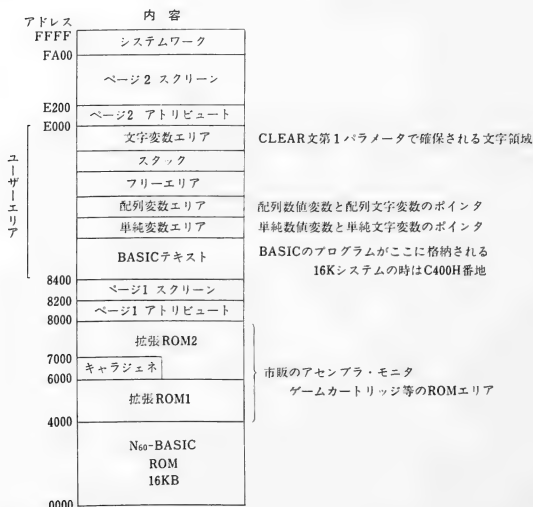


図2-1 メモリ・マップ

2-2 ユーザーエリア



図2-2 ユーザーエリア

ユーザープログラムの先頭アドレスは拡張 RAM がない場合、C401H、拡張した場合、8401H になります。またユーザーエリアは使用するページ数で異なりますが、7100バイト～30140バイトの間となります。

それではプログラム入力によって各ポイントがどう変わるか調べてみますが、市販のモニタ ROM をお持ちでない方のために BASIC・モニタプログラムを載せておきます。ただし、BASIC プログラムは各ポイントを変化させるので、さらに詳しく学習されたい方はモニタ ROM の購入をお勧めします(付-8 参照)。

BASIC のポイントの説明にあたって、ひとつ注意することがあります。以下のポイント値は、キーボード入力時のもので、カセットから LOAD した場合、同じプログラムでも、異なる値を示すことがあります。

これは、N₆₀-BASIC ではカセットでの LOAD、SAVE 時に、余分の 00 が入るためで、詳しくは第 6 章で述べることにします。


```

100 REM カサベン
110 DIM C(12)
120 INPUT "a= "; A
130 INPUT "b= "; B: C(0)=A*B
140 PRINT "a×b="; C(0)
150 END
Ok
RUN
a= ? 256
b= ? 16
a×b= 4096
Ok

```

上記のサンプルプログラムで、その入力前、プログラム入力後、実行後のポインタの変化を調べてみましょう。

	ポインタ アドレス	電源ON時 (RESET)	プログラム 入力後	プログラム 実行後	備 考
プログラム 開始アドレス	FA5F,60	8401] プログラム領域] 単純変数領域] 配列変数領域] 文字列領域 (50バイト)
変数領域 の始まり	FF56,57	8403	8458		
配列変数領域 の始まり	FF58,59	8403	8458	8466	
フリーエリア の始まり	FF5A,5B	8403	8458	84AE	
スタック の始まり	FA5B,5C	DFCD			
文字列領域 の始まり	FF27,28	DFFF			

図 2-3 各ポインタの変化

注：RAM32KBでページ2を指定した時の値です

2-3 プログラムの格納状態

次のサンプルプログラムがメモリ内にどのように格納されているか説明します。

[illegible]

アドレス	データ	内容
8400	プログラム先頭	100 REM カケザン
8401	0D	
8402	84	840D... リンクポインタ (次の行の開始アドレス)
8403	64	
8404	00	0064... 行番号 (10進数に変換すると100)
8405	8E	REMの中間言語
8406	20	スペース
8407	B6	*カ*
8408	B9	*ケ*
8409	BB	*サ*
840A	DE	*//*
840B	DD	*ン*
840C	00	行の終わりを示す
840D	19	
840E	84	8419... リンクポインタ 110 DIM C(12)
840F	6E	
8410	00	006E... 行番号
8411	85	DIMの中間言語
8412	20	スペース
8413	43	*C* 変数名
8414	28	*(* 左カッコ
8415	31	*1*
8416	32	*2* 配列の数(ASCIIコード)
8417	29	*) 右カッコ
8418	00	行の終わり



図2-4 プログラムの格納状態

リンクポインタが0000Hになったアドレスがプログラムテキストの終了アドレスと解釈され、その次のアドレスが変数領域の始まりとなります。

プログラムの格納のされ方は、PC-8001 の N-BASIC とだいたい同じですが、細部で異なる部分があります。

(1) GOTO, GOSUB, THEN の後に続く行番号

(2) 数値

この2つが、N-BASIC と比べて大幅に変わっています。

(1)の飛び先の行番号は N₈₀-BASIC では、行番号が ASCII コードで入っています。そして、その ASCII コードで入っている行番号を2バイトの16進数に変換し、現在の行番号より小さければプログラムの先頭から、行番号が大きければ、現在の行の後から、一致する行番号を探しますので、飛び先の行番号によって処理する時間が変わります。

LIST

```
10 GOTO 100
20 END
100 PRINT "100"
110 END
Ok
```

それでは上記のプログラムで、実行前と実行後での行番号の変化を調べてみましょう。

N-BASIC の場合

実行前

GOTO の 行番号を示す 10進数に
中間コード 識別コード 直すと100

```
8020 00 2B 80 0A 00 89 20 0E 64 00 00 31 80 14 00 81
8030 00 3D 80 64 00 91 20 22 31 30 30 22 00 43 80 6E
8040 00 81 00 00 00
```

実行後

アドレスを示す
識別コード

リンクポインタ 行番号(100)

100行のあるアドレス

```
8020 00 2B 80 0A 00 89 20 0D 30 80 00 31 80 14 00 81
8030 00 3D 80 64 00 91 20 22 31 30 30 22 00 43 80 6E
8040 00 81 00 00 00
```

N₆₀-BASIC の場合

実行前

GOTO の
中間言語 "1" "0" "0" — ASCII コード

```
400 00 0B 84 0A 00 88 20 31 30 30 00 11 84 14 00 80
410 00 1D 84 64 00 95 20 22 31 30 30 22 00 23 84 6E
420 00 80 00 00 00
```

実行後

変化なし

```
8400 00 0B 84 0A 00 88 20 31 30 30 00 11 84 14 00 80
8410 00 1D 84 64 00 95 20 22 31 30 30 22 00 23 84 6E
8420 00 80 00 00 00
```

このように N₆₀-BASIC では飛び先を毎回 1 行ごとに調べていきますので GOTO, GOSUB を多用したプログラムは実行速度が遅くなります。

また、数値の格納のされ方は、N-BASIC と N₆₀-BASIC ではまったく異質のものとなっています。これは N-BASIC が数値として整数、単精度、倍精度が使用できるために、その型に合わせてメモリに格納されることに起因しています。

例として A に 1234567890 を代入したときの N-BASIC と N₆₀-BASIC との違いを調べてみます。

N60-BASIC の場合

```
10 a=1234567890:print a
LIST
10 A=1234567890:PRINT A
Ok
RUN
1. 23456789E+09
Ok
```

N-BASIC の場合

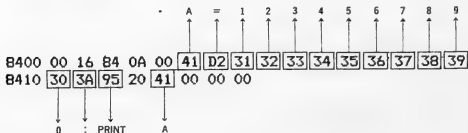
```
10 a=1234567890:print a
list
10 A=1234567890#:PRINT A
Ok
run
1.23457E+09
Ok
```

N₆₀-BASIC では、入力した値と LIST したときの値は同じになっています。メモリ内部も入力した値がそのまま入っていて、変数Aに数値を代入するときに5バイトの浮動小数点表記に変換されます。このため例のように、代入する数値が10桁以上の場合は、指数表示になります。

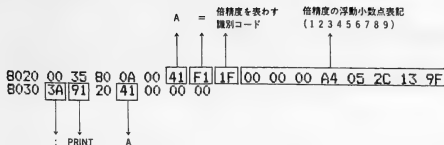
N-BASIC の場合6桁以上だと、とくに指定していない限り倍精度扱いになります。そのため LIST したときに倍精度数値を表わす"#"が数値の後ろに付き、代入する変数Aが単精度のときは、倍精度から単精度へ変換され6桁の指数表示となります。

これらの変化をメモリ上で調べてみましょう。

N60-BASIC の場合



N-BASIC の場合



2-4 プログラムの回復のさせ方

BASIC のプログラムを誤って消すことがあります。この原因としては

1. 電源スイッチを切った、電源プラグが抜けた、停電した。
2. NEW を実行した。
3. RESET ボタンを押した。

等が考えられますが、1については絶対に元には戻りません。2については回復可能です。問題は3の場合です。プログラムが7100バイト以下ならば回復可能ですが、もし7100バイト以上のときは回復しません。これは RESET をかけるとイニシャライズ(初期設定)プログラムが働くためです。このときにページ1～4までをモード1の状態にセットします。それによって 8000～83FFH, A000～A3FFH, C000～C3FFH, E000～E3FFH の VRAM の内容が書き換えられます。もし、この部分に BASIC のプログラムがあった場合、書き換えられてしまいます。このようになってあわてるよりも 頻繁に CSAVE しておくことをお勧めします。

では、2の場合の回復の仕方について説明しましょう。

先ほどのプログラムを NEW してみます。

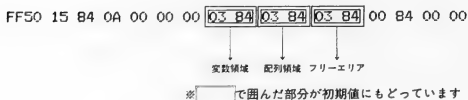
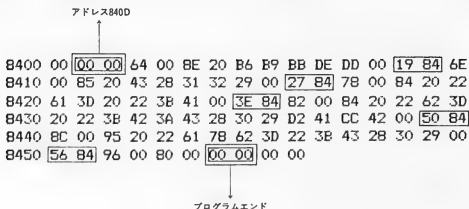
LIST

```

100 REM カササ*ン
110 DIM C(12)
120 INPUT "a= "; A
130 INPUT "b= "; B: C(0)=A*B
140 PRINT "a×b="; C(0)
150 END
Ok
new
Ok

```

メモリ格納状態



メモリダンプから分かるように最初のリンクポインタの値が0000Hになっており、変数領域等のポインタの値が初期値に戻っています。これらのポインタ値を戻せばプログラムが復活するわけです。

ここでこれらのポインタの戻し方について説明します。

1. 8405H 番地(16KBのときはC405H)から1バイトずつ読み出して00を書き込んであるアドレスを探し出す。
2. そのアドレス+1の値を下位、上位の順に 8401,02H 番地に格納する(これがリンクポインタです)。
3. この時のリンクポインタを使ってエンドマーク(0000H)を見つける。
4. 見つかったアドレス+2の値を下位、上位の順に FF56,57H 番地にセットする。

以上の処理を行なうことによってプログラムが回復します。

この一連の処理を BASIC でプログラミングするのは無意味なので、機械語で行なってみました。

アドレス	マシン語	ニーモニック
0000	2104B4	LD HL, 8404H
0003	23	LOOP1: INC HL
0004	7E	LD A, (HL)
0005	B7	OR A
0006	20FB	JR NZ, LOOP1
0008	23	INC HL
0009	2201B4	LD (8401H), HL
000C	23	LOOP2: INC HL
000D	23	INC HL
000E	2256FF	LD (0FF56H), HL
0011	2B	DEC HL
0012	7E	LD A, (HL)
0013	B7	OR A
0014	C8	RET Z
0015	2B	DEC HL
0016	6E	LD L, (HL)
0017	67	LD H, A
0018	1BF2	JR LOOP2

このプログラムはリロケートブルになっていますので任意の番地で使用できます。

例 DE00H 番地から入力する。

例 DE00 番地のとき

```
DE00 21 04 B4 23 7E B7 20 FB 23 22 01 B4 23 23 22 56
DE10 FF 2B 7E B7 C8 2B 6E 67 1B F2 FF FF FF FF FF FF
```

EXEC&Hde00

Ok

LIST

```
100 REM カササヘン
110 DIM C(12)
120 INPUT "a= "; A
130 INPUT "b= "; B: C(0)=A*B
140 PRINT "a*b="; C(0)
150 END
Ok
```


2-5 中間言語

N₆₀-BASIC では命令(キーワード)は、メモリ節約、処理速度向上のため、中間言語と呼ばれる 1 バイトのコードで格納されます。

```
100 INPUT A
110 C=SQR(A)
120 PRINT C
```

```
8400 00 09 84 64 00 84 20 41 00 14 84 6E 00 43 D2 DC
8410 28 41 29 00 1C 84 78 00 95 20 43 00 00 00 00
```

※ _____ で示される部分が中間言語です

命令 (KEYWORD)	中間言語
INPUT	84
=	D2
SQR	DC
PRINT	95

図2-5 キーワードと中間言語の対応

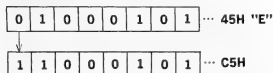
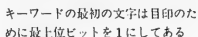
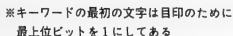
キーワードは 021D~036DH 番地までデータとして ASCII コードで入っています。



最初の文字は最上位ビットが 1 になっている

図 2-6 データの形式

それでは実際に格納されているメモリ内容を調べてみましょう。



N₆₀-BASIC の命令(キーワード)を出力するプログラムを次に紹介します。

次に、キーワードと中間言語の対応を調べるプログラムを示します。

```

10 AD=&H21D:FOR I=&H80 TO &HAA
20 DA=I:GOSUB 200:PRINT HE$:TAB(5);
30 A=PEEK(AD):PRINT CHR$(A AND &H7F);:AD=AD+1
40 A=PEEK(AD):IF A>12? THEN 60
50 PRINT CHR$(A);:AD=AD+1:GOTO 40
60 PRINT TAB(14):CHR$(I):NEXT I
100 FOR I=&HC2 TO &HF1
110 DA=I:GOSUB 200:PRINT HE$:TAB(5);
120 A=PEEK(AD):PRINT CHR$(A AND &H7F);:AD=AD+1
130 A=PEEK(AD):IF A>12? THEN 150
140 PRINT CHR$(A);:AD=AD+1:GOTO 130
150 PRINT TAB(14):CHR$(I):NEXT I
160 END

```

```

200 REM hex$
210 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 230
220 HE=DA-(16*HF):GOSUB 230:HE$=RIGHT$(HE$,2):RE
TURN
230 IF HE>9 THEN HE=HE+55:GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE):RETURN

```

80	END	全	CA	+	ハ
81	FOR	●	CB	-	ヒ
82	NEXT	●	CC	*	フ
83	DATA	●	CD	/	ヘ
84	INPUT	○	CE	^	ホ
85	DIM	●	CF	AND	マ
86	READ	を	D0	OR	ミ
87	LET	あ	D1	>	ム
88	GOTO	い	D2	=	メ
89	RUN	う	D3	<	モ
8A	IF	ま	D4	SGN	ヤ
8B	RESTORE	お	D5	INT	ユ
8C	GOSUB	や	D6	ABS	ヨ
8D	RETURN	ゆ	D7	USR	ラ
8E	REM	よ	D8	FRE	リ
8F	STOP	っ	D9	INP	ル
90	OUT		DA	LPOS	レ
91	ON	あ	DB	POS	ロ
92	LPRINT	い	DC	SQR	ワ
93	DEF	う	DD	RND	ン
94	POKE	え	DE	LOG	ッ
95	PRINT	お	DF	EXP	。たち
96	CONT	き	E0	COS	つて
97	LIST	く	E1	SIN	と
98	LLIST	け	E2	TAN	な
99	CLEAR	こ	E3	PEEK	に
9A	COLOR	さ	E4	LEN	め
9B	PSET	し	E5	HEX\$	わ
9C	PRESET	す	E6	STR\$	のは
9D	LINE	せ	E7	VAL	ひ
9E	PAIN	そ	E8	ASC	ふ
9F	SCREEN		E9	CHR\$	へ
A0	CLS		EA	LEFT\$	ほ
A1	LOCATE	。r	EB	RIGHT\$	ま
A2	CONSOLE	」	EC	MID\$	み
A3	CLOAD	、	ED	POINT	む
A4	CSAVE	・	EE	CSRLIN	
A5	EXEC	ヲ	EF	STICK	
A6	SOUND	フ	F0	STRIG	
A7	PLAY	アイ	F1	TIME	
A8	KEY	ウ			
A9	LCOPY	エ			
AA	NEW	ツ			
C2	TABC	テ			
C3	TO	ト			
C4	FN	ナ			
C5	SPCC	ニ			
C6	INKEY\$	ヌ			
C7	THEN	ネ			
C8	NOT	ノ			
C9	STEP				

下上	8	9	A	B	C	D	E	F
0	♠ END	OUT	CLS	-	タ	ミ	た	み
1	♥ FOR	あ ON	え LOCATE	ア	チ	ム	ち	む
2	♣ NEXT	い LPRINT	「 CONSOLE	イ	ツ	メ	つ	め
3	♦ DATA	う DEF	」 CLOAD	ウ	テ	モ	て	も
4	○ INPUT	え POKE	、 CSAVE	エ	ト	ヤ	と	や
5	● DIM	お PRINT	・ EXEC	オ	ナ	ユ	な	ゆ
6	を READ	か CONT	ヲ SOUND	カ	ニ	ヨ	に	よ
7	あ LET	き LIST	ア PLAY	キ	ヌ	ラ	ぬ	ら
8	い GOTO	く LLIST	イ KEY	ク	ネ	リ	ね	り
9	う RUN	け CLEAR	ウ LCOPY	ケ	ノ	ル	の	る
A	え IF	こ COLOR	エ NEW	コ	ハ	レ	は	れ
B	お RESTORE	さ PSET	オ	サ	ヒ	ロ	ひ	ろ
C	や GOSUB	し PRESET	ヤ	シ	フ	ワ	ふ	わ
D	ゆ RETURN	す LINE	ユ	ス	ヘ	ン	へ	ん
E	よ REM	せ PAINT	ヨ	セ	ホ	、	ほ	
F	つ STOP	そ SCREEN	ッ	ソ	マ	°	ま	
					AND	EXP	STICK	

図2-9 中間言語表(コード順)

2-6 中間言語処理ルーチン

BASIC プログラムを RUN させると N₆₀-BASIC インタプリタは中間言語に対応したジャンプテーブルを参照して処理先のアドレスを求めます。このジャンプテーブルはワークエリア上にあります。

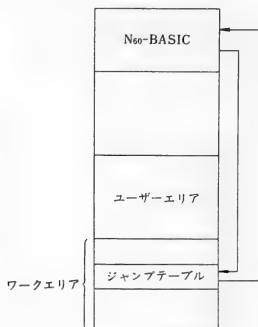


図 2-10 ジャンプテーブルの位置

このジャンプテーブルは各 2 バイトで未使用分を含めて、110個(220バイト)あります。これらのジャンプテーブルは 2つのブロックに分かれており、第 1 ブロックがコマンド・ステートメント用、第 2 ブロックが関数用になっています。

第 1 ブロックが、66個(132バイト)で FA61~FAE4H 番地を使用しており、中間言語の 80H~C1H に対応しています。第 2 ブロックは関数専用で 44個(88バイト)あり、FAE5~FB3CH 番地を使用しており、D4~FFH までの中間言語が割り付けられています。

AB~C1H、F2~FFH までの中間言語に対応する命令は N₆₀-BASIC にはありませんが、将来、拡張 BASIC 等で使用されると思われます。

この、ジャンプテーブルを使用して N₆₀-BASIC の命令追加、機能強化をすることができます。

さて、これらのジャンプテーブルの値は 0195~021CH 番地にあり、イニシャライズの時に RAM 上に転送されます。

中間言語コードに対応する処理アドレスを求めるプログラムは次のとおりです。

```

10 AD=&HFA60:FOR I=&H80 TO &HAA
20 DA=I:GOSUB 200:PRINT HE$:TAB(10):AD=AD+2
30 DA=PEEK(AD):GOSUB 200:PRINT HE$:AD=AD-1
40 DA=PEEK(AD):GOSUB 200:PRINT HE$:AD=AD+1
50 NEXT I
100 AD=&HFAE4:FOR I=&HD4 TO &HEC
110 DA=I:GOSUB 200:PRINT HE$:TAB(10):AD=AD+2
120 DA=PEEK(AD):GOSUB 200:PRINT HE$:AD=AD-1
130 DA=PEEK(AD):GOSUB 200:PRINT HE$:AD=AD+1
140 NEXT I
150 END
199 REM hex$
200 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 220
210 HE=DA-(16*HF):GOSUB 220:HE$=RIGHT$(HE$,2):RE
TURN
220 IF HE>9 THEN HE=HE+55:GOTO 240
230 HE=HE+&H30
240 HE$=HE$+CHR$(HE):RETURN

```

80	3535	A8	2353
81	067E	A9	22A6
82	35F7	AA	34CD
83	07E0	D4	3898
84	09AB	D5	39E7
85	3302	D6	38B9
86	0A09	D7	0755
87	07F5	D8	32DE
88	07A0	D9	0DCC
89	0781	D9	0DCC
8A	0861	DA	0D22
8B	3519	DB	0D27
8C	078F	DC	3F92
8D	07BC	DD	3BA3
8E	07E2	DE	3EA5
8F	3533	DF	3E21
90	0DD6	E0	3F51
91	0844	E1	3F57
92	087A	E2	3FD3
93	0D3A	E3	0DF3
94	0DFA	E4	3229
95	087E	E5	03EA
96	356B	E6	305B
97	05DB	E7	32BA
98	05D6	E8	3238
99	35A9	E9	3249
9A	1D9B	EA	3257
9B	2D3C	EB	3286
9C	2D37	EC	328F
9D	2DC7		
9E	2EDC		
9F	1E04		
A0	1DF8		
A1	1CD2		
A2	1CF6		
A3	2496		
A4	247E		
A5	261D		
A6	1E9B		
A7	1EB3		

対応表を見て分かるように、中間言語で C2～D3H, ED～F1H の処理アドレスがありません。これらはインタプリタ内部で特殊処理用に使われています。

NOT, FN, INKEY\$ および POINT～TIME(ED～F1H)はコンペア(比較)命令によってインタプリタ内でチェックされており、個別にジャンプします。

また、TAB, TO, SPC, THEN, STEPは、この命令のみで使われることはありません。たとえば、SPC, TAB は、必ず PRINT 文と一緒に使われるので、PRINT の処理ルーチンの中で、TAB と SPC の処理を行なっています。

残りの命令は演算子であり、これらも式の中で使われますので、式解析ルーチンの中でジャンプさせています。

これらのジャンプ先は次のようになっています。

C2	0926
C3	
C4	0D61
C5	0926
C6	2771
C7	
C8	0CF9
C9	
CA	367E
CB	3683
CC	37B4
CD	3803
CE	3EFA
CF	0C9A
D0	0C99
D1	
D2	
D3	
ED	2D55
EE	0D30
EF	2236
F0	2256
F1	1E83

2-7 識別コード

N-BASICにおいては、数値の型を区別するために 0B~1FH まで識別コードとして使っていますが、単精度や倍精度がない N₆₀-BASIC では、数値を区別するための識別コードがなくなっています。N₆₀-BASIC で識別コードとして使用しているのは 14H で、これはキャラクタコードの 00~1FH のグラフィックキャラクタのために使われています。この部分は本来、コントロールコードになっており、グラフィックキャラクタとコントロールコードとの区別をつけるために、識別子が用いられています。

実際にグラフィックキャラクタがどのように入っているか調べてみましょう。

10 PRINT "月火水木金土日年月時"

8400	00	1E	84	0A	00	95	20	22	14 31	14 32	14 33	14 34
8410	14 35	14 36	14 37	14 38	14 39	14 3A	22	00	00	00		

"月"は 14H+31H の 2 バイトで入っています。これで分かるように 14H に続くキャラクタコードには 30H のポリュウムを持たせています。

このため"月"を画面に CHR\$ で表示するとき

PRINT CHR\$(1)

では表示されずに

PRINT CHR\$(&H14) ; CHR\$(&H31)

で表示されます。

2-8 単純変数領域

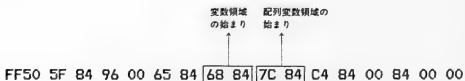
単純変数領域は、プログラム領域の後ろより、始まります。プログラムを実行して実際にその変数が使われますと、この変数領域に登録されます。この領域は、FF56, 57H で示されるアドレスから、(FF58, 59H で示されるアドレス)-1 までになります。

実際に単純変数の状態を調べましょう。

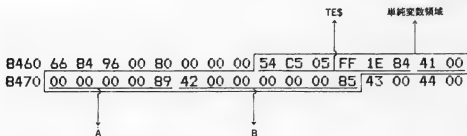
```

100 REM カケザン
110 DIM C(12):TES="カケザン"
120 INPUT "A= ";A
130 INPUT "B= ";B:C(0)=A*B
140 PRINT TES,"A×B=";C(0)
150 END
Ok
RUN
A= ? 256
B= ? 16
カケザン      A×B= 4096
Ok

```



このプログラムでは単純変数領域は 8468~847BH 番地までになっており、その格納のされ方は次のようになります。



このプログラムでは単純変数としては TES, A, B の 3 つがありました。それぞれメモリ内では、

変数名 変数の文字数 文字列の格納されているポインタ
 TE\$ 8468 54 C5 05 FF 1E 84 (841EHより文字列があることを示している)
 "T" "E"
 ↓
 bit7を1にしてある
 A 846E 41 00 00 00 00 00 89
 変数名 数値255を表わす(浮動小数点表記)
 "A"
 B 8475 42 00 00 00 00 00 85 変数Aと同じ

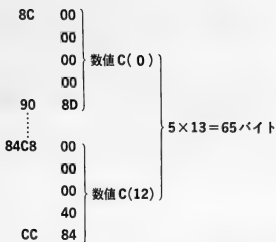
のように格納されています。

N₆₀-BASICでは変数の種類が数値と文字との2つしかありませんので、文字型変数の変数名の2文字目をその区別に使用しており、最上位ビット(bit7)を1にすることによって文字型であることを示しています。

	変数名	数値またはディスクリプタ部分	バイト数
数値変数	2 バイト	5 バイト	7 バイト
文字変数	2 バイト	文字数 ダミー ポインタ ※ スtringディスクリプタ	6 バイト

※ ポインタ文字列の格納先アドレスを示す

図 2-11 単純変数の格納のされ方



次に、配列の次元数が多次元の場合について調べてみましょう。

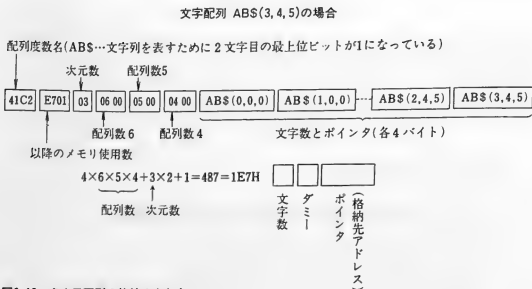
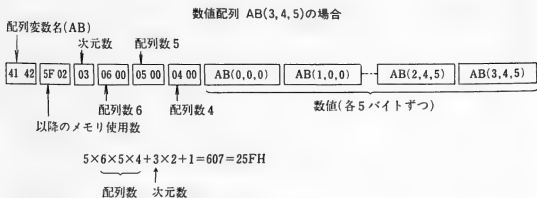


図2-12 多次元配列の格納のされ方

2-10 文字変数と文字列領域

変数が数値の場合は、その値にかかわらず5バイトの浮動小数点表記で表わされ、メモリ数に変化がありませんが、文字列のときは、0～255バイトまでの文字データが許されるために、文字列によってメモリ数が変化します。このため N₈₀-BASIC では、変数が文字列の場合、ストリングディスクリプタと呼ばれる4バイトのデータが変数領域に格納されます。この格納のされ方は2-8、2-9で説明したとおりです。

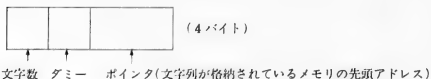


図 2-13 ストリングディスクリプタの構成

文字列は、文字列領域に格納され、ポインタもこの領域を示しますが、文字列が、プログラム中にあるときに限って、ポインタはプログラムエリアを示します。このときは当然のこととして、文字列領域には格納されません。この処理があることがメモリの節約になっています。

```
100 A$="PC-6001"  
110 FOR I=0 TO 2:READ NM$(I):NEXT  
120 DATA System,Soft,Fukuoka  
130 INPUT "Phone No. ";PH$  
Ok  
RUN  
Phone No. ? 092-714-6354  
Ok
```

それでは、このプログラム実行後の変数のポインタを調べてみましょう。

まず文字列領域(CLEARの第1パラメータで指定)はFA5B, 5CHで示されるアドレス+1からFF27, 28Hで示されるアドレスまでです。

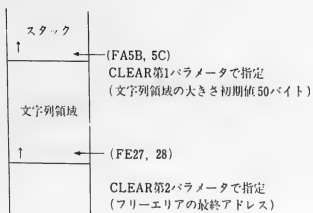
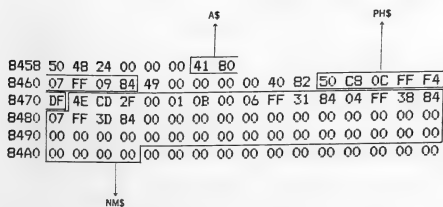


図2-14 文字列領域



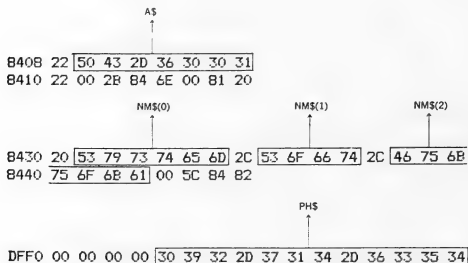
DFCE~DFFFH までの50バイトが文字列領域になっているのが分かります。次に文字変数を調べてみます。



変 数 名	stringディスクリプタ のアドレス	stringディスクリプタ		文字列格納場所
		文 字 数	ポインタ	
AS	8 4 6 0	7	8 4 0 9	プログラム領域
PHS	8 4 6 D	12	D F F 4	文字列領域
NMS(0)	8 4 7 8	6	8 4 3 1	プログラム領域
NMS(1)	8 4 7 C	4	8 4 3 8	"
NMS(2)	8 4 8 0	7	8 4 3 D	"

図 2-15 文字変数のディスクリプタ

格納場所のポインタを調べてみます。



PHS は文字列領域の終わりの方から格納されていきます。文字列データが変わった場合や、新しい文字変数が使用されたときは、前へと格納されていきます。そして文字列領域いっぱいになると、不要となった文字列データをつめていきます。これをガベージコレクションと呼んでおり、このためにプログラムによってはしばらくの間、実行が中断することがあります。

2-11 フリーエリア

フリーエリアを調べる命令として、FRE 関数があります。プログラムエリアの残りを調べるときは()の内に数値を入れ、文字列領域の残りを知りたいときは文字変数を使います。

32KB ページ 2 のとき

```
How Many Pages? 2
N60-BASIC
By Microsoft (c) 1981
23484 Bytes free
Ok
?fre(0)
23484
Ok
?fre(a$)
50
Ok
```

16KB ページ 2 のとき

```
How Many Pages? 2
N60-BASIC
By Microsoft (c) 1981
7100 Bytes free
Ok
?fre(0)
7100
Ok
?fre(a$)
50
Ok
```

文字列領域はページ数、増設 RAM の有無に関係なく、初期値は50バイトになります。
N60-BASIC 起動時のページ数とフリーエリアのバイト数の関係は次のようになります。

RAM	ページ数	フリーエリア	文字列領域 ^④	ユーザーメモリ上限 ^⑤
32KB	1	3 0 1 4 0	5 0	F 9 F F H
	2	2 3 4 8 4	5 0	D F F F H
	3	1 5 2 9 2	5 0	B F F F H
	4	7 1 0 0	5 0	9 F F F H
16KB	1	1 3 7 5 6	5 0	F 9 F F H
	2	7 1 0 0	5 0	D F F F H

④ CLEARの第1パラメータで指定される

⑤ CLEARの第2パラメータで指定される

図 2-16 ページ数とフリーエリア

これらフリーエリアに関するアドレスは、次のようになります。

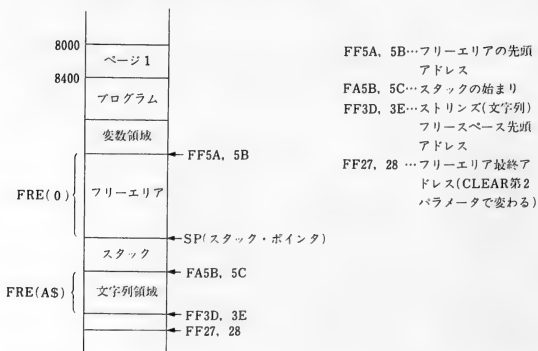


図2-17 フリーエリアに関するアドレス

2-10で使ったプログラムを実行して、そのフリーエリアの変化を調べてみましょう。

32K システム ページ 2 で起動

```
100 A$="PC-6801"
110 FOR I=0 TO 2:READ NM$(I):NEX
T
120 DATA System,Soft,Fukuoka
130 INPUT "Phone No. ";PH$
Ok
RUN
Phone No. ? 092-714-6254
Ok
PRINT fre(0),fre(a$)
23323 38
Ok
```

プログラム実行後のポインタ

```
FF58 71 84 A4 84 44 84 00 00
FA58 01 0E 00 CD DF FF FF 01
```

FF38 FF 07 FF B0 40 F3 DF 27

FF20 FF FF FF 00 00 01 00 FF DF 00 B4 2D FF 07 FF B0

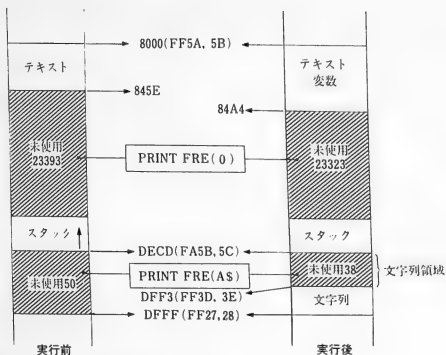


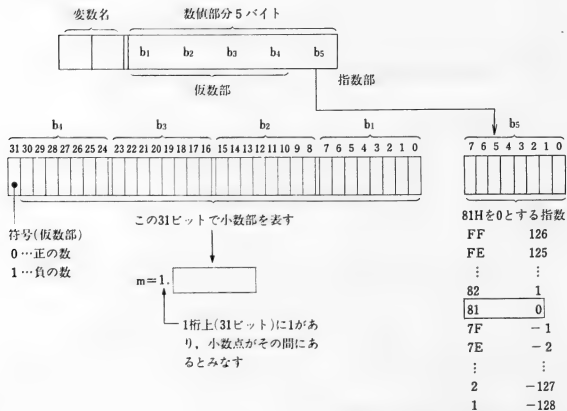
図2-18 フリーエリアの変化

実行後、フリーエリア(プログラムエリアの残り)が70バイト ($23393 - 23323 = 70$)、文字列領域が12バイト ($50 - 38 = 12$)減っています。フリーエリアの70バイトの内訳は、 $6 + 7 + 51 + 6 = 70$ になります。文字列領域の方は、PH\$ の"092-714-6254"の12バイトが文字列領域に書き込まれています。

2-12 浮動小数点表記法

変数領域に格納される数値および、演算は5バイトの浮動小数点表記法で行なわれています。浮動小数点表記では、数値は仮数部+指数部で表現されます。

格納状態を図に表わすと次のようになります。



指数部が0の時は数値を0とみなす
(数値が0の時は特殊処理により仮数部の値を無視する)

図2-19 浮動小数点の表記法

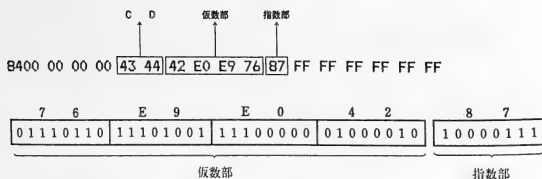
これにより数値 n をもとめる式は、

$$n = lm \times 2^r$$

となります。

実際に数値を入れてみましょう。

cd=123.456789
0k



$$\downarrow$$

$$+ 1.1110110111010011110000001000010 \times 2^6$$

図2-20 指数部と仮数部の表現法

この値を計算するには、

$$\begin{aligned}
 &+ 1, 111011011 \dots 0010 \\
 &\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
 &(2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-5} + \dots + 2^{-30}) \times 2^6 \\
 &\downarrow \\
 &(1 + 0.5 + 0.25 + 0.125 + 0.03125 \dots + 0.0000000001) \times 64
 \end{aligned}$$

上記の方法では計算に時間がかかり過ぎますので、簡単な方法を次に示します。

$$\begin{aligned}
 &(1 + 76E9E042H / 2^{31}) \times 2^6 \\
 &\quad \downarrow \\
 &(1 + 1995038786 / 2147483648) \times 64 \\
 &\quad \downarrow \\
 &(1 + 0.92901233) \times 64 = 123.456789
 \end{aligned}$$

となり代入値と等しいことが分かります。

第3章 CRTディスプレイ

- 3-1 VDG(ビデオディスプレイジェネレータ)
- 3-2 アトリビュート
- 3-3 VRAMのアドレス
 - 3-3-1 アトリビュートアドレスマップ
 - 3-3-2 テキスト・セミグラフィックアドレスマップ
 - 3-3-3 グラフィックアドレスマップ
- 3-4 キャラクタ・ジェネレータ
- 3-5 ページの切り換え
- 3-6 表示期間

第3章 CRT ディスプレイ

3-1 VDG(ビデオディスプレイジェネレータ)

(1) CRT コントローラの IC として M5C6847P-1(MC6847 と同等品)が使用されています。この VDG は家庭用 TV に接続するのを目的としてつくられたディスプレイ用の IC です。

機能としては次のようなものがあります。

イ.32文字×16行(512文字)のアルファニューメリック表示。

ロ.グラフィック表示モード。

ハ.64文字のキャラクタ・ジェネレータ内蔵。

VDG は 4 種類のアルファニューメリック表示と 8 種のグラフィック表示の機能を持っており、各モードの一覧を図 3-1 に示します。

このモードの内、N₆₀-BASIC では外部アルファニューメリック、セミグラフィック 6, 128×192カラーグラフィック、256×192グラフィックモードをサポートしています。





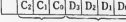
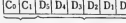

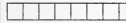








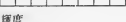
表示(1画面256×192ドット)		デ ー タ	モ ー ド	N60- BASIC のモード
モード	内容(数字はドット単位の長さ)			
32文字 ×16行	 12 5×7ドット1文字	 ASCIIコード	内部アルファ ニューメリック	—
32文字 ×16行	 128×12ドット1文字	 外部キャラジュネのアドレス	外部アルファ ニューメリック	1
64×32 ドット	 4画素は同色	 カラー 輝度	セミグラフィック4	—
64×48 ドット	 6画素は同色	 カラー 輝度	セミグラフィック6	2
64×64 ドット	 E3 E2 E1 E0	 E3 E2 E1 E0	64×64 カラーグラフィック	—
128×64 ドット	 D7 D6 D5 D4 D3 D2 D1 D0	 輝度	128×64 グラフィック	—
28×64 ドット	 E3 E2 E1 E0	 E3 E2 E1 E0	128×64 カラーグラフィック	—
128×96 ドット	 D7 D6 D5 D4 D3 D2 D1 D0	 輝度	128×96 グラフィック	—
128×96 ドット	 E3 E2 E1 E0	 E3 E2 E1 E0	128×96 カラーグラフィック	—
128×192 ドット	 D7 D6 D5 D4 D3 D2 D1 D0	 輝度	128×192 グラフィック	—
128×192 ドット	 E3 E2 E1 E0	 E3 E2 E1 E0	128×192 カラーグラフィック	3
256×192 ドット	 D7 D6 D5 D4 D3 D2 D1 D0	 輝度	256×192 グラフィック	4

図3-1 全モード一覧

3-2 アトリビュート

アトリビュート(属性)エリアに格納されているデータによって、カラーおよび表示モードを設定します。アトリビュートエリアは512(32×16)バイト使用しており、そのアドレスは、図3-2で示されるとおりです。

ページ	RAM16KB	RAM32KB
1	C 0 0 0 ~ C 1 F F H	8 0 0 0 ~ 8 1 F F H
2	E 0 0 0 ~ E 1 F F H	E 0 0 0 ~ E 1 F F H
3		C 0 0 0 ~ C 1 F F H
4		A 0 0 0 ~ A 1 F F H

図3-2 アトリビュートエリア

アトリビュートメモリの出力は VDG のコントロール信号に接続されており、このアトリビュートの値を変えることによってモードを変更します。

各コントロール信号は次のように接続されています。

7	6	5	4	3	2	1	0 データビット
\overline{A}/G	\overline{A}/S	\overline{INT}/ENT	GM0	GM1	GM2	CSS	INV

図3-3 コントロール信号の接続

- INV** アルファニューメリック時、バックと文字の色を変える。
- CSS** カラーセットセレクト入力(色相を180°変えて色を変える)。
- GM0~GM2** グラフィックモード切り換え。
- \overline{INT}/ENT**
 - キャラクター・モード時CGを内部または外部の選択。
 - 0=内部
 - 1=外部
 - セミグラフィック時セミグラフィック4モードか6モードの選択。
 - 0=4モード
 - 1=6モード
- \overline{A}/S** アルファニューメリックかセミグラフィックの切り換え。
- 0=アルファニューメリック
- 1=セミグラフィック
- \overline{A}/G** アルファニューメリックとグラフィックの切り換え。
- 0=アルファニューメリック
- 1=グラフィック

アトリビュートの初期値は次のようになります。

モード1…20H モード3…8CH

モード2…60H モード4…DCH

これを出力するプログラムを紹介します。

アトリビュートを調べるプログラム

```
10 FOR I=1 TO 4:SCREEN I,2,2:CLS:A(I)=PEEK(&HE000):NEXT I:SCREEN 1,1,1
20 FOR I=1 TO 4:DA=A(I):GOSUB 200:PRINT "モード "; I; "=";HE$;"H":NEXT I
30 END
200 REM hex$
210 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 230
220 HE=DA-(16*HF):GOSUB 230:HE$=RIGHT$(HE$,2):RETURN
230 IF HE>9 THEN HE=HE+55:GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE):RETURN
```

実行結果

```
RUN
モード 1= 20H
モード 2= 60H
モード 3= 8CH
モード 4= DCH
Ok
```

次に各コントロール記号を説明します。

(1) INV(bit0)

テキスト画面のみに働き、文字とバックの色を反転させます。

```
10 SCREEN 1,2,2:COLOR 1:CLS
20 FOR I=32 TO 255:PRINT CHR$(I);:NEXT I
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A OR 1:NEXT I
```

上記のプログラムを実行すると、緑のバックに白の文字であったのが、白のバックに緑の文字に変化します(実際は白色はやや黄色を帯びて表示される場合もあります)。

これは"COLOR 2"を実行したのと同じ結果になります。

(2) CSS(bit1)

モード1～4まですべてに対して機能し、色相を180°変えることによって、色を変えることができます。これは COLOR 命令の3番目のパラメータに対応しています。

```
10 SCREEN 1,2,2:COLOR 1:CLS
20 FOR I=32 TO 255:PRINT CHR$(I):NEXT I
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A OR
  2:NEXT I
```

上記のプログラムを実行すると分かるように、バックを緑からオレンジへ変化させます。

(3) GM0～GM2(bit4～2)

グラフィックのモードを切り換えます。グラフィックモードにするには \overline{A}/G 信号を“1”にする必要があります。

GM 0	GM 1	GM 2	グラフィックモード
0	0	0	64×64カラーグラフィック
1	0	0	128×64グラフィック
0	1	0	128×64カラーグラフィック
1	1	0	128×96グラフィック
0	0	1	128×96カラーグラフィック
1	0	1	128×192グラフィック
0	1	1	128×192カラーグラフィック
1	1	1	256×192グラフィック

図3-4 グラフィックモードの切り換え

(4) \overline{INT}/ENT (bit5)

この信号は2つの使い方があり、キャラクタ・モード(テキストモード)では、VDG 内部の CG(キャラクタ・ジェネレータ)を使用するか、外部の CG を使用するかを選択に使います。また、セミグラフィックモードでは、4モードと6モードの切り換えに用いられています。

まず、キャラクタ・モードについて説明しましょう。

PC-6001 では外部 CG を使用していますので、このビットは“1”になりますが、“0”にすることによって内部 CG に切り換えることができます。

```
10 SCREEN 1,2,2:COLOR 1:CLS
20 FOR I=32 TO 255:PRINT CHR$(I):NEXT I
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A AND
  D &HDF:NEXT I
```

このプログラムを実行すると、画面の文字が小さくなっていきます。ところがカナやグラ

フィックのキャラクタであった所が英文字や英記号を表示してしまいます。これは内部キャラクタが64文字(6bit ASCII)しかないために、ASCII コードに対応するイメージが発生してこのようになったわけです。

データ (H)	キャラ クタ	データ (H)	キャラ クタ	データ (H)	キャラ クタ	データ (H)	キャラ クタ
00	@	10	P	20	SP	30	0
01	A	11	Q	21	!	31	1
02	B	12	R	22	"	32	2
03	C	13	S	23	#	33	3
04	D	14	T	24	\$	34	4
05	E	15	U	25	%	35	5
06	F	16	V	26	&	36	6
07	G	17	W	27	,	37	7
08	H	18	X	28	(38	8
09	I	19	Y	29)	39	9
0A	J	1A	Z	2A	*	3A	:
0B	K	1B	[2B	+	3B	;
0C	L	1C	\	2C	,	3C	<
0D	M	1D	[2D	-	3D	=
0E	N	1E	↑	2E	.	3E	>
0F	O	1F	←	2F	/	3F	?

図3-5 VDG内部キャラクタ

次にセミグラフィック時の4モードおよび6モードについて調べます。

N₈₀-BASICでは、6モード(64×48セミグラフィックモード)をサポートしています。次のプログラムを実行して直線の太さをよく見て下さい。

```
10 SCREEN 2,2,2:COLOR 1:CLS
20 LINE(0,93)-(256,93),2:LINE(0,0)-(256,192),3
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A AND
  D &HDF:NEXT I
```

直線が太くなったのが分かると思います。6モードでは1点は4×4ドットでしたが、4モードでは4×6ドットになるために、2ドット分だけ線が太くなったのです。

(5) $\bar{A}/S(\text{bit}6)$

アルファニューメリックとセミグラフィックの切り換えに使われます。"0"のときにアルファニューメリック(モード1)になり、"1"でセミグラフィック(モード2)になります。

(6) $\bar{A}/G(\text{bit}7)$

アルファニューメリックとグラフィックの切り換えに用いられます。“0”のときにはアルファニューメリックモード(モード1およびモード2)、“1”でグラフィックモード(モード3、4)になります。

3-3 VRAM のアドレス

3-3-1 アトリビュートアドレスマップ

アトリビュートエリアは、モードに関係なく512バイトに固定されています。
アドレスを表にすると次のようになります。

行 \ 桁	0	1	2	3	4	5	6	7	8	9		23	24	25	26	27	28	29	30	31
0	X000	X001	X002	X003	X004	X005	X006	X007	X008	X009	X017	X018	X019	X01A	X01B	X01C	X01D	X01E	X01F
1	X020	X021	X022	X023	X024	X025	X026	X027	X028	X029	X037	X038	X039	X03A	X03B	X03C	X03D	X03E	X03F
2	X040	X041	X042	X043	X044	X045	X046	X047	X048	X049	X057	X058	X059	X05A	X05B	X05C	X05D	X05E	X05F
3	X060	X061	X062	X063	X064	X065	X066	X067	X068	X069	X077	X078	X079	X07A	X07B	X07C	X07D	X07E	X07F
4	X080	X081	X082	X083	X084	X085	X086	X087	X088	X089	X097	X098	X099	X09A	X09B	X09C	X09D	X09E	X09F
5	X0A0	X0A1	X0A2	X0A3	X0A4	X0A5	X0A6	X0A7	X0A8	X0A9	X0B7	X0B8	X0B9	X0BA	X0BB	X0BC	X0BD	X0BE	X0BF
6	X0C0	X0C1	X0C2	X0C3	X0C4	X0C5	X0C6	X0C7	X0C8	X0C9	X0D7	X0D8	X0D9	X0DA	X0DB	X0DC	X0DD	X0DE	X0DF
7	X0E0	X0E1	X0E2	X0E3	X0E4	X0E5	X0E6	X0E7	X0E8	X0E9	X0F7	X0F8	X0F9	X0FA	X0FB	X0FC	X0FD	X0FE	X0FF
8	X100	X101	X102	X103	X104	X105	X106	X107	X108	X109	X117	X118	X119	X11A	X11B	X11C	X11D	X11E	X11F
9	X120	X121	X122	X123	X124	X125	X126	X127	X128	X129	X137	X138	X139	X13A	X13B	X13C	X13D	X13E	X13F
10	X140	X141	X142	X143	X144	X145	X146	X147	X148	X149	X157	X158	X159	X15A	X15B	X15C	X15D	X15E	X15F
11	X160	X161	X162	X163	X164	X165	X166	X167	X168	X169	X177	X178	X179	X17A	X17B	X17C	X17D	X17E	X17F
12	X180	X181	X182	X183	X184	X185	X186	X187	X188	X189	X197	X198	X199	X19A	X19B	X19C	X19D	X19E	X19F
13	X1A0	X1A1	X1A2	X1A3	X1A4	X1A5	X1A6	X1A7	X1A8	X1A9	X1B7	X1B8	X1B9	X1BA	X1BB	X1BC	X1BD	X1BE	X1BF
14	X1C0	X1C1	X1C2	X1C3	X1C4	X1C5	X1C6	X1C7	X1C8	X1C9	X1D7	X1D8	X1D9	X1DA	X1DB	X1DC	X1DD	X1DE	X1DF
15	X1E0	X1E1	X1E2	X1E3	X1E4	X1E5	X1E6	X1E7	X1E8	X1E9	X1F7	X1F8	X1F9	X1FA	X1FB	X1FC	X1FD	X1FE	X1FF

“X” はページによって異なります。

ページ	RAM 32KB	RAM 16KB
1	8	C
2	E	E
3	C	
4	A	

図 3-6 ページと X の値

3-3-2 テキスト・セミグラフィックアドレスマップ

これもアトリビュートと同じで32文字×16行で512バイトになります。

行 \ 桁	0	1	2	3	4	5	6	7	8	9		23	24	25	26	27	28	29	30	31
0	X200	X201	X202	X203	X204	X205	X206	X207	X208	X209	X217	X218	X219	X21A	X21B	X21C	X21D	X21E	X21F
1	X220	X221	X222	X223	X224	X225	X226	X227	X228	X229	X237	X238	X239	X23A	X23B	X23C	X23D	X23E	X23F
2	X240	X241	X242	X243	X244	X245	X246	X247	X248	X249	X257	X258	X259	X25A	X25B	X25C	X25D	X25E	X25F
3	X260	X261	X262	X263	X264	X265	X266	X267	X268	X269	X277	X278	X279	X27A	X27B	X27C	X27D	X27E	X27F
4	X280	X281	X282	X283	X284	X285	X286	X287	X288	X289	X297	X298	X299	X29A	X29B	X29C	X29D	X29E	X29F
5	X2A0	X2A1	X2A2	X2A3	X2A4	X2A5	X2A6	X2A7	X2A8	X2A9	X2B7	X2B8	X2B9	X2BA	X2BB	X2BC	X2BD	X2BE	X2BF
6	X2C0	X2C1	X2C2	X2C3	X2C4	X2C5	X2C6	X2C7	X2C8	X2C9	X2D7	X2D8	X2D9	X2DA	X2DB	X2DC	X2DD	X2DE	X2DF
7	X2E0	X2E1	X2E2	X2E3	X2E4	X2E5	X2E6	X2E7	X2E8	X2E9	X2F7	X2F8	X2F9	X2FA	X2FB	X2FC	X2FD	X2FE	X2FF
8	X300	X301	X302	X303	X304	X305	X306	X307	X308	X309	X317	X318	X319	X31A	X31B	X31C	X31D	X31E	X31F
9	X320	X321	X322	X323	X324	X325	X326	X327	X328	X329	X337	X338	X339	X33A	X33B	X33C	X33D	X33E	X33F
10	X340	X341	X342	X343	X344	X345	X346	X347	X348	X349	X357	X358	X359	X35A	X35B	X35C	X35D	X35E	X35F
11	X360	X361	X362	X363	X364	X365	X366	X367	X368	X369	X377	X378	X379	X37A	X37B	X37C	X37D	X37E	X37F
12	X380	X381	X382	X383	X384	X385	X386	X387	X388	X389	X397	X398	X399	X39A	X39B	X39C	X39D	X39E	X39F
13	X3A0	X3A1	X3A2	X3A3	X3A4	X3A5	X3A6	X3A7	X3A8	X3A9	X3B7	X3B8	X3B9	X3BA	X3BB	X3BC	X3BD	X3BE	X3BF
14	X3C0	X3C1	X3C2	X3C3	X3C4	X3C5	X3C6	X3C7	X3C8	X3C9	X3D7	X3D8	X3D9	X3DA	X3DB	X3DC	X3DD	X3DE	X3DF
15	X3E0	X3E1	X3E2	X3E3	X3E4	X3E5	X3E6	X3E7	X3E8	X3E9	X3F7	X3F8	X3F9	X3FA	X3FB	X3FC	X3FD	X3FE	X3FF

"X" の値はページによって異なります。

ページ	RAM 32KB	RAM 16KB
1	8	C
2	E	E
3	C	
4	A	

図 3-7 ページと X の値

3-3-3 グラフィックアドレスマップ

グラフィックモードでは32バイト×192ラインで6144バイトのメモリを使用しています。

	0	1	2	3	4	5	6	7	8	9		23	24	25	26	27	28	29	30	31
0	0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	0217	0218	0219	021A	021B	021C	021D	021E	021F
1	0220	0221	0222	0223	0224	0225	0226	0227	0228	0229	0237	0238	0239	023A	023B	023C	023D	023E	023F
2	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0257	0258	0259	025A	025B	025C	025D	025E	025F
3	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0277	0278	0279	027A	027B	027C	027D	027E	027F
4	0280	0281	0282	0283	0284	0285	0286	0287	0288	0289	0297	0298	0299	029A	029B	029C	029D	029E	029F
5	02A0	02A1	02A2	02A3	02A4	02A5	02A6	02A7	02A8	02A9	02B7	02B8	02B9	02BA	02BB	02BC	02BD	02BE	02BF
6	02C0	02C1	02C2	02C3	02C4	02C5	02C6	02C7	02C8	02C9	02D7	02D8	02D9	02DA	02DB	02DC	02DD	02DE	02DF
7	02E0	02E1	02E2	02E3	02E4	02E5	02E6	02E7	02E8	02E9	02F7	02F8	02F9	02FA	02FB	02FC	02FD	02FE	02FF
8	0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	0317	0318	0319	031A	031B	031C	031D	031E	031F
9	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0337	0338	0339	033A	033B	033C	033D	033E	033F
10	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0357	0358	0359	035A	035B	035C	035D	035E	035F
11	0360	0361	0362	0363	0364	0365	0366	0367	0368	0369	0377	0378	0379	037A	037B	037C	037D	037E	037F
12	0380	0381	0382	0383	0384	0385	0386	0387	0388	0389	0397	0398	0399	039A	039B	039C	039D	039E	039F
13	03A0	03A1	03A2	03A3	03A4	03A5	03A6	03A7	03A8	03A9	03B7	03B8	03B9	03BA	03BB	03BC	03BD	03BE	03BF
14	03C0	03C1	03C2	03C3	03C4	03C5	03C6	03C7	03C8	03C9	03D7	03D8	03D9	03DA	03DB	03DC	03DD	03DE	03DF
15	03E0	03E1	03E2	03E3	03E4	03E5	03E6	03E7	03E8	03E9	03F7	03F8	03F9	03FA	03FB	03FC	03FD	03FE	03FF
	⋮										⋮									
181	18A0	18A1	18A2	18A3	18A4	18A5	18A6	18A7	18A8	18A9	18B7	18B8	18B9	18BA	18BB	18BC	18BD	18BE	18BF
182	18C0	18C1	18C2	18C3	18C4	18C5	18C6	18C7	18C8	18C9	18D7	18D8	18D9	18DA	18DB	18DC	18DD	18DE	18DF
183	18E0	18E1	18E2	18E3	18E4	18E5	18E6	18E7	18E8	18E9	18F7	18F8	18F9	18FA	18FB	18FC	18FD	18FE	18FF
184	1900	1901	1902	1903	1904	1905	1906	1907	1908	1909	1917	1918	1919	191A	191B	191C	191D	191E	191F
185	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1937	1938	1939	193A	193B	193C	193D	193E	193F
186	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1957	1958	1959	195A	195B	195C	195D	195E	195F
187	1960	1961	1962	1963	1964	1965	1966	1967	1968	1969	1977	1978	1979	197A	197B	197C	197D	197E	197F
188	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1997	1998	1999	199A	199B	199C	199D	199E	199F
189	19A0	19A1	19A2	19A3	19A4	19A5	19A6	19A7	19A8	19A9	19B7	19B8	19B9	19BA	19BB	19BC	19BD	19BE	19BF
190	19C0	19C1	19C2	19C3	19C4	19C5	19C6	19C7	19C8	19C9	19D7	19D8	19D9	19DA	19DB	19DC	19DD	19DE	19DF
191	19E0	19E1	19E2	19E3	19E4	19E5	19E6	19E7	19E8	19E9	19F7	19F8	19F9	19FA	19FB	19FC	19FD	19FE	19FF

ライン

実際に使用する場合はページによってアドレスの補正を行なう必要があります。

例 RAM32KB でページ 2 のとき

実際のアドレス = E000H + 表のアドレス

補正値

ページ 2 = E000H

ページ 3 = C000H

ページ 4 = A000H

3-4 キャラクタ・ジェネレータ

キャラクタ・ジェネレータ ROM は、アルファニューメリックモードでは CG(キャラクタ・ジェネレータ)として、グラフィックモード時では、グラフィックパターン ROM として使用されています。この CG ROM は OUT 命令によって ROM のバンクを切り換えて読み出すことが可能で、CG ROM の1文字あたり16バイト分ある内の12バイトを使用して1文字を構成しています。たとえば" B "の文字は次のようになっています。

ADR	76543210	DB
6420H		00
6421H		7C
6422H		22
6423H		22
6424H		22
6425H		3C
6426H		22
6427H		22
6428H		22
6429H		7C
642AH		00
642BH		00
Ok		

8ドット

12ドット

これを出力するプログラムは下記のとおりです。

```

10 OUT &H93, 4: AD=&H6000+16*&H42
20 PRINT "ADR 76543210 DB"
30 FOR I=AD TO AD+11: DA=INT(I/256): GOSUB 200: PR
  INT HE$:
40 DA=I-INT(I/256)*256: GOSUB 200: PRINT HE$: "H "
  ;
50 A=PEEK(I): DA=A: FOR J=7 TO 0 STEP -1: IF A>=2^
  J THEN COLOR 2: A=A-2^J
60 PRINT " ": COLOR 1: NEXT J: PRINT " ": GOSUB
  200: PRINT HE$: NEXT I
70 OUT&H93, 5: END
200 REM HEX$
210 HE$="00": HE=INT(DA/16): HF=HE: GOSUB 230
220 HE=DA-(16*HF): GOSUB 230: HE$=RIGHT$(HE$, 2): RE
  TURN
230 IF HE>9 THEN HE=HE+55: GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE): RETURN

```

なお、OUT&H93, 4でCG ROM を6000H 番地から読むことができ、OUT&H93, 5でCG を OFF にします。

キャラクタ・フォントを出力するプログラムを紹介しておきます。


```

10 OUT &H93,4:FOR K=0 TO 255:AD=&H6000+16*K
20 DA=INT(AD/256):GOSUB 200:PRINT HE$:DA=AD-INT(AD/256)*256:GOSUB200
30 PRINT HE$:DA=K:GOSUB 200:PRINT"h ";HE$;" ";
40 IF K<32 THEN PRINT CHR$(&H14);CHR$(K+&H30);"
   ";GOTO 60
50 PRINT CHR$(K);" ";
60 FOR I=AD TO AD+5:DA=PEEK(I):GOSUB 200:PRINT
HE$;" ";NEXTI:PRINT
70 FOR I=AD+6 TO AD+11:DA=PEEK(I):GOSUB 200:PRINT
TAB(12);HE$;" ";NEXT
80 PRINT:NEXT K:OUT&H93,5:END
200 REM HEX$
210 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 230
220 HE=DA-(16*HF):GOSUB 230:HE$=RIGHT$(HE$,2):RETURN
230 IF HE>9 THEN HE=HE+55:GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE):RETURN

```

CG は完全にデコードされていないために、7000～7FFFH まではイメージが出力されます。

3-5 ページの切り換え

出力ポートの B0H の bit1, 2 が画面の切り換えのポートになっており、この値を変えることによってページを切り換えることができます。

bit 2	bit 1	VRAMアドレス	RAM32KB 時のページ	RAM16KB 時のページ
0	0	C 0 0 0 H	3	1
0	1	E 0 0 0 H	2	2
1	0	8 0 0 0 H	1	
1	1	A 0 0 0 H	4	

図 3-8 ページの切り換えポート

出力ポートに出力するだけで画面を切り換えられますので、これをうまく使えば画面の擬似重ね合わせができます。

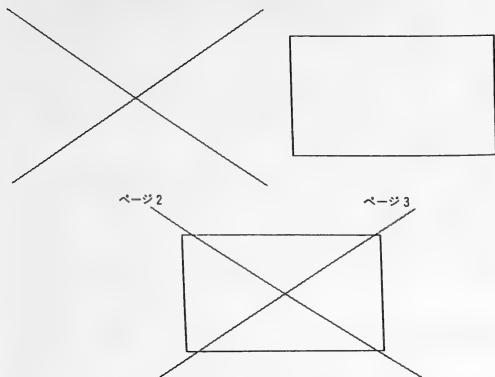
プログラム例

```

10 SCREEN 4,2,2:CLS
20 LINE(0,0)-(256,192):LINE(256,0)-(0,192)
30 SCREEN 4,3,3:CLS
40 LINE(30,30)-(220,160),,B
50 OUT&HB0,0:OUT&HB0,2:GOTO50

```

実行例 このように画面の重ね合わせが可能

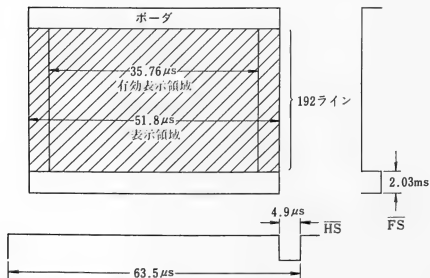


ページ 2 とページ 3 を 50 行で切り換えていますので少しチラつきますが、×と□が重なって見えるはずで(注：このプログラムを実行するにはページの指定が 3 または 4 になっていなければなりません)。

ただし、この重ね合わせは、グラフィックどうし、またはアルファニューメリックモードどうしならば問題はありませんが、グラフィックとアルファニューメリックの組み合わせでは、画面の同期がとれません。(逆にこれをゲーム等に利用するとおもしろいと思いますが)。

3-6 表示期間

画面にノイズを出さないようにするために、表示期間中は DMA(ダイレクト・メモリ・アクセス)を行ない CPU を停止させており、帰線消去の間のみ、メモリのリード、ライトを許可しています。このために、CPU にかかなりのロスタイムが生じています。



※斜線部分でCPUが停止する

図3-9 画面表示期間

第4章 キー入力

- 4-1 ファンクションキー
 - 4-1-1 メモリの格納状態
 - 4-1-2 ROM内の格納状態
 - 4-1-3 KEY LIST
 - 4-1-4 内容の定義の仕方
 - 4-1-5 キーポインタとファンクションキーフラグの使い方
- 4-2 キー入力ステートメント
 - 4-2-1 INPUT
 - 4-2-2 INKEY\$
 - 4-2-3 STICK, STRIG
 - 4-2-4 キーバッファ
- 4-3 コントロールキー

第4章 キー入力

4-1 ファンクションキー

4-1-1 メモリの格納状態

ファンクションキーの初期設定は次のようになっています。

COLOR	SCREEN
CLOAD"	CSAVE"
GOTO	PRINT
LIST	PLAY
RUN	CONT

これらの内容は、ワークエリア上の FB3DH 番地から FB8CH 番地に格納されています。

```
FB3D 43 4F 4C
FB40 4F 52 20 00 00 43 4C 4F 41 44 22 00 00 47 4F 54
FB50 4F 20 00 00 00 4C 49 53 54 20 00 00 00 52 55 4E
FB60 0D 00 00 00 00 53 43 52 45 45 4E 20 00 43 53 41
FB70 56 45 22 00 00 50 52 49 4E 54 20 00 00 50 4C 41
FB80 59 20 00 FF FF 43 4F 4E 54 0D 00 00 00 FF FF 13
```

このように1つのファンクションキーに対して8バイト使用されており、8文字に満たない場合はエンドマークとして「00」が書き込まれます。

8文字で完全に管理されているために、PC-8001のように隣のあいているファンクションキー領域を使用し、9文字以上定義することはできません。

4-1-2 ROM 内の格納状態

ファンクションキーの内容はROM内の0167H番地から017AH番地に中間言語+キャラクタ(1文字)の2バイトで入っており、初期設定のときにFA33~FA46Hに転送されます。

そして中間言語をキーワードに変換してファンクションキー・バッファに入れます。

```

F 1  0167  9A 20  COLOR+sp
F 2  0169  A3 22  CLOAD+"
F 3  016B  88 20  GOTO+sp
F 4  016D  97 20  LIST+sp
F 5  016F  89 0D  RUN+cr
F 6  0171  9F 20  SCREEN+sp
F 7  0173  A4 22  CSAVE+"
F 8  0175  95 20  PRINT+sp
F 9  0177  A7 20  PLAY+sp
F10  0179  96 0D  CONT+cr

```

	RAM	ROM
F 1	FA33	0167
F 2	FA35	0169
F 3	FA37	016B
F 4	FA39	016D
F 5	FA3B	016F
F 6	FA3D	0171
F 7	FA3F	0173
F 8	FA41	0175
F 9	FA43	0177
F 10	FA45	0179

図 4-1 キー内容の格納アドレス

プログラム中でファンクションキーのイニシャライズをするときは、KEY コマンドよりは機械語を使う方が簡単です。その方法は第 9 章で述べます。

4-1-3 KEY LIST

画面にはファンクションキーの内容表示が 5 文字までしかなされません。N₆₀-BASIC では内容をすべて表示する KEY LIST の命令がありませんので、ファンクションキーの内容を見たいときはそれぞれのキーを押すしか方法がありません。

BASIC でキーの内容を表示するプログラムを下記に示します。

```

10 I=0
20 K=I:GOSUB 100
30 PRINT TAB(10);:K=I+5:GOSUB 100:PRINT
40 I=I+1:IF I=5 THEN END
50 GOTO 20
100 FOR J=0 TO 7
110 A=PEEK(&HFB3D+(8*K+J))
115 IF A=0 THEN 140
120 IF A<32 THEN A=32
130 PRINT CHR$(A);:NEXT J
140 RETURN

```

4-1-4 内容の定義の仕方

ファンクションキーの定義は以下に行ないます。

ファンクションキー定義の例

```
key 1, "ABCDE"
Ok
key 2, "12345"+chr$(13)
Ok
a$="AAAAAA":key 3, a$
Ok
key 4, chr$(28)+chr$(29)+chr$(30)
+chr$(31)
Ok
key 5, chr$(34)+"UFO"+chr$(34)+chr
$(13)
Ok
```

1 ABCDE 12345 AAAAAA "UFO"

ファンクションキーにはコントロールコードも定義できますが、画面のウィンドウには表示されません。

```
key 1, chr$(18)+chr$(18)+chr$(18)
+chr$(18)
Ok
key 2, chr$(30)+chr$(30)+chr$(30)
+chr$(30)
Ok
key 3, chr$(8)+chr$(1)+chr$(2)+chr
$(3)
Ok
key 4, "load"+chr$(13)
Ok
key 5, "list."+chr$(30)+chr$(30)
Ok
```

1 load list.

PC-6001のファンクションキー

1 load list.

PC-8001のファンクションキー

PC-8001のファンクションキーは、PC-6001と同様です。

4-1-5 キーポイントとファンクションキーフラグの使い方

ファンクションキーフラグを使うと、ファンクションキーが押されたのと等価の働きをします。この機能を使うと、いろいろとおもしろいことができます。

```
10 KEY 1,"LIST"+CHR$(13)
20 POKE &HFB8D,&H3D:POKE &HFB8E,&HFB
30 POKE &HFA32,5
```

```
RUN
Ok
LIST
```

```
10 KEY 1,"LIST"+CHR$(13)
20 POKE &HFB8D,&H3D:POKE &HFB8E,
&HFB
30 POKE &HFA32,5
Ok
```

キーポインタは FB8DH と FB8EH で、キーフラグは FA32H です。

キーフラグには、キー内容の文字数を設定します。上記のプログラムでは、ファンクションキーの内容が、"LIST"+0DH の5文字なので30行で POKE する値が5になっています。

これから分かるように、キーフラグに設定できる文字数は最大255文字になります。

ただし、文字列の中に00が入っていると、そこで文字列の終了と解釈されます。

4-2 キー入力ステートメント

N60-BASIC には、キー入力ステートメントが INPUT と INKEY\$ の 2 種類しかありません。この 2 つを比較してみましょう。

		INPUT		INKEY\$
書 式		INPUT A, INPUTA\$		A\$ = INKEY\$
入力表示	プロンプト (" ")	可能		不可
	入力指示の ? 表示	有		無
	カーソル表示	有		無
	入力時のカーソル移動	有		無
	エコーバック	有		無
	入力待ち	待つ		待たない
キー入力	入力文字の種類	英数, カタカナ ヒラガナ, ファンクション キー, グラフィックキー	A 数字のみ	A\$ 可
		特殊文字	カンマは変数の区切り	
		コントロールコードの入力	不可	
		入力文字数	39桁以内	71文字以内
	実行	RET キー		自動的
	入力文字なしで RET キー	前に入力した値		CHR\$(13)
	使用できる画面モード	1 または 2		全モード可
	STOP キー	中断		中断
入力中断	CTRL + C	中断		中断
	BREAK時のBEEP音	音がでる		音がでる
	CONTによるプログラム再開	可		可

図4-2 キー入力比較

4-2-1 INPUT

INPUT 命令はテキストモードでしか使用できません。たとえばページ 2 でモード 3 のときに INPUT 命令があると、ページ 1, モード 1 でキー入力になります。

```
例: 10 SCREEN 3, 2, 2
    20 INPUTA$
```

そのためモード3, 4においてINPUTを使うようなプログラムはつくれないことになり
ますので思考型のゲーム等をつくる場合は注意が必要です。

これを解決する方法としては INKEY\$ を使って複数の文字を入力するようにプログラム
を組みばよいでしょう。

4-2-2 INKEY\$

INKEY\$ コマンドを使うと、キー入力を持たないで次の処理に進めますのでリアルタイム
処理向きといえます。

また、IF 文で判断させて、文字に対応した行に飛ばすのが一般的な利用方法です。

例 トランプゲームのとき

```
1000 PRINT "トノ カート ヲ ステマスカ (1-5)"
1010 A$=INKEY$: IF A$="1" THEN 1100
1020 IF A$="2" THEN 1200
1030 IF A$="3" THEN 1300
1040 IF A$="4" THEN 1400
1050 IF A$="5" THEN 1500
1060 GOTO 1010
```

INKEY\$ を使ってモード3, 4で文字列を入力する方法を示します。

INKEY\$ による文字列の入力

```
10 SCREEN 3,2,2:CLS
20 A$=""
30 B$=INKEY$: IF B$="" THEN 20
40 B=ASC(B$): IF B=13 THEN 100
50 A$=A$+B$:PRINT B$:
60 GOTO 20
100 END
```

4-2-3 STICK, STRIG

PC-8001のリアルタイムゲームのほとんどは、テンキーとスペースキーを使って遊ぶよう
になっています。PC-8001の場合は、I/O ポートを INP 命令で調べることによって、簡単
に、押されたキーを知ることができますが、PC-6001はキーボードをサブ CPU が管理して
いるためにこの方法が使えません。それではゲームで遊ぶとき不便なので STICK と STRIG
の命令があります。

STICK はジョイスティック(カーソルキー)の方向を調べる命令で、STRIG は発射ボタン
(スペースキー)が押されているか調べます。ジョイスティックは2人まで遊べるようにと、2
個同時につながるようになっています。

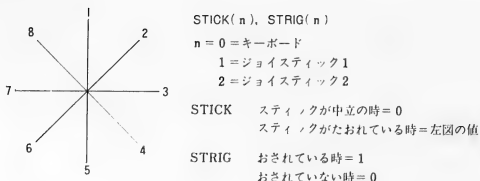


図4-3 ジョイスティックの方向とSTICK・STRIG

STICK, と STRIG を機械語で使用する時の方法を説明します。

```

0000 3E00      STICK: LD   A,0
0002 CD3922    CALL  2239H
0005 CD4107    CALL  0741H
0008 7B        LD    A,E
0009 C9^       RET

```

```

0000 3E00      STRIG: LD   A,0
0002 CD5922    CALL  2259H
0005 CD4107    CALL  0741H
0008 7B        LD    A,E
0009 C9        RET

```

Acc(アキュムレータ)にスティックの番号(00~02H)をセットして、サブルーチンをコールします。データがFAC(浮動小数点アキュムレータ)にセットされていますので、0741HをコールしてFACの値をDEレジスタペアにセットし、Eの値をAccに移します。この部分は第9章でも説明してあります。

また、実際にゲームなどで使用する場合は、スティックの方向より、どのスイッチが押されているかが分かったほうが使いやすいでしょう。その方法を簡単に説明します。

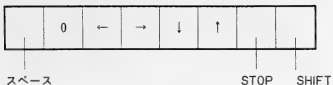
●スティック 0

```

CALL 1061H
RET

```

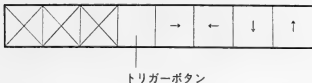
結果が次の図のように Acc に返ってきます。



各キーを押しているとそのビットが1になる。

●スティック 1, 2

```
LD A, STICKNO ; スティックの番号 1 または 2
CALL 1CA6H
RET
```



4-2-4 キーバッファ

キースキャンはサブ CPU が担当しており、キー入力があるたびに割り込みをかけてメイン CPU に送ります。受け取ったデータを一時キーバッファに格納し、INPUT 待ちになったときにキーバッファより取り出します。そのため、メイン CPU が他の処理をしている間に次の処理をキー入力することができます。

先行キー入力の実験

```
10 CLEAR 300:CLS
20 FOR I=1 TO 400:LOCATE0,0:PRINT I:NEXT I
30 INPUT A$:PRINT A$
```

上のプログラムを実行すると画面の左上に数字が表示されます。このときに何かキーを押して下さい。数字が400までカウントすると30行の INPUT 命令を実行します。このときに先ほどキー入力した文字が画面に表示されます。この先行入力機能は便利なのですが、ゲームなどでは非常に使いづらいものとなります。

なぜなら、ゲーム等でキー操作を行なっている場合、たいていの人はいはしばらくキーを押したままにしますので、キーリピートの機能が働き、そのキーのデータが、連続してキーバッファに取り込まれてしまうのです。

INKEY\$を使っている、ポーカー、マージャンなどのいくつかのゲームの中には、この影響をもらに受けて、同じカードやパイを連続して捨ててしまうなどの現象を起こすものがあります。

キーバッファは、FBB9H より、64バイト分あり、FB8F~FB94H がこのバッファの管理行なっています。この管理部分のポインタをクリアすれば良いわけです。

次のプログラムは、キーバッファがクリアされる例です。

4-3 コントロールキー

PC-8001 はコントロールコードのシンボルを表示することができますが、PC-6001 では、その部分がグラフィックキャラクタになっているために表示することができません。

$$D_E \begin{matrix} & S_1 & S_2 & E_1 & E_2 & E_3 & E_4 & E_5 & E_6 & E_7 & E_8 & E_9 & E_{10} & E_{11} & E_{12} & E_{13} & E_{14} & E_{15} & E_{16} \\ D_1 & D_1 & D_2 & D_3 & D_4 & D_5 & D_6 & D_7 & D_8 & D_9 & D_{10} & D_{11} & D_{12} & D_{13} & D_{14} & D_{15} & D_{16} & D_{17} & D_{18} \end{matrix} \rightarrow \leftarrow \uparrow \downarrow$$

元 月 大 水 木 金 土 日 年 時 小 分 秒 百 千 万
上 下 十 十 十 十 一 一 一 一 一 一 一 一 一 一 一

これらのコントロールコードは、前述の PRINT CHR\$ の書式で使う他に、**CTRL** キーと、アルファベットの組み合わせで入力することができます。

16進	10進	シンボル	シンボルの意味	対応するキー	機 能
00	0		null		
01	1	SH	Start of Heading(ヘッディング開始)	CTRL + A	
02	2	SX	Start of Text(テキスト開始)	" + B	カーソルを1項目ごとに左へ移す。
03	3	EX	End of Text(テキスト終了)	" + C	STOPと同01プログラムの実行を中止して、N60-BASICのメニュー画面に戻る。
04	4	ET	End of Transmission(伝送終了)	" + D	
05	5	EQ	Enquiry(問い合わせ)	" + E	カーソル位置から後ろをすべて消す。
06	6	AK	Acknowledge(肯定応答)	" + F	カーソルを1項目ごとに右にずらす。
07	7	BL	Bell(ベル、ブザー)	" + G	内蔵のブザーを0.5秒間鳴らす。
08	8	BS	Back Space(後退)	" + H	DEL カーソルのすぐ左の1文字を除する。
09	9	HT	Horizontal Tabulation(水平タブ)	" + I	8文字ごとの水平タイプ。
0A	10	LF	Line Feed(改行)	" + J	行を2つに分ける。カーソルを次の行へ移す。
0B	11	HM	Home(VT)Vertical Tabulation(垂直タブ)	" + K	カーソルをホームポジション画面左上に戻す。
0C	12	CL	Clear(FF)Form Feed(改頁)	" + L	画面を消去してカーソルをホームポジションに戻す。
0D	13	CR	Carriage Return(復帰)	" + M	カーソルを次の行の先頭に移す。
0E	14	SO	Shift-out(シフトアウト)	" + N	
0F	15	SI	Shift-in(シフトイン)	" + O	
10	16	DE	Data Link Escape(伝送制御拡張)	" + P	
11	17	D1	Device Control1(装置制御1)	" + Q	
12	18	D2	Device Control2(装置制御2)	" + R	INS
13	19	D3	Device Control3(装置制御3)	" + S	
14	20	D4	Device Control4(装置制御4)	" + T	
15	21	NK	Negative Acknowledge(否定応答)	" + U	1行消去
16	22	SN	Synchronous idle(同期信号)	" + V	
17	23	EB	End of Transmission Block(伝送ブロック終了)	" + W	
18	24	CN	Cancel(取消し)	" + X	
19	25	EM	End of Medium(媒体終端)	" + Y	
1A	26	SB	Substitute(文字置換)	" + Z	
1B	27	EC	Escape(拡張)	ESC	実行の一時中断
1C	28	→	(FS)File Separator(ファイル分離)		カーソルを1つ右へ移す。
1D	29	←	(GS)Group Separator(グループ分離)		カーソルを1つ左へ移す。
1E	30	↑	(RS)Record Separator(レコード分離)		カーソルを上への行へ移す。
1F	31	↓	(US)Unit Separator(ユニット分離)		カーソルを下への行へ移す。

図4-4

第5章 サウンド機能

- 5-1 PLAY命令の基礎
- 5-2 SOUND命令
- 5-3 より高度なテクニック
 - 5-3-1 PLAYのバッファ
 - 5-3-2 PLAYに変数を
 - 5-3-3 PLAYと音色
 - 5-3-4 サウンド機能と機械語

第5章 サウンド機能

5-1 PLAY 命令の基礎

サウンド機能は PC-6001 を特徴づける機能の 1 つです。この章ではこのサウンド機能について解説しますが、その初めとして PLAY 命令について説明しましょう。

N₆₀-BASIC の PLAY 命令は音楽を演奏するための命令で、大変分かりやすい音楽用言語 (MML: Music Macro Language) を使って音程、音量、音長等のデータを表記します。

音の高さは 2 種類の表記方法があり、第 1 の方法は音階名 (C, D, E, F, G, A, B) に On, およびシャープ (# または +)・フラット (−) を併用する方法、もう 1 つは Nn を使う方法です。

まず音階名による表記とは、出したい音の音階名を A~G のアルファベットで列記する方法です。図 5-1 を参考にしながら次の命令を実行してみてください。

PLAY "CDEFGABC"

いかがですか？ ドレミファソラシドと演奏しましたね。ところが最後のドは最初のドと同じ高さの音が出てしまいました。これはオクターブの指定をしなかったために起きたこと



図5-1

です。PLAY 命令では、C～Bまでを1オクターブとして区切り、計8オクターブの音域をカバーしています。そしてO1～O8をMML群に付け加えることによって、そのオクターブを指定することができます。オクターブの指定は次に新たにオクターブを指定するまで保持されます。それでは先ほどのものにオクターブの指定を付けてみます。

PLAY "O4CDEFGABO5C"

今度は正しく1オクターブ上のドが出ました。

今度は半音を出してみましょう。半音を出す場合は、半音にしたい音階名の後にシャープの記号(#か+)やフラットの記号(、マイナス記号)を付ければよいのです。試しに音を出してみましょう。

PLAY "CC#D"

ドとレの音の間にド#が出ました。これは次のように書いても同じです。

PLAY "CC#D"

PLAY "CD-D"

上の例はPLAY 命令中では+と#が同じ意味をもつためです。下の例はド#とレ♯が同じ音だからです。分からない人は音楽の本を見てください。ですからミ#つまりE+はファ、つまりFと同じ音になりますし、ド♯つまりC-はシ、つまりBと同じ音になります。ただし、この場合、本来の音より1オクターブ上のC-が出ます(このあたりはある程度、音楽の基礎知識が必要ですから、音楽の苦手な方は読みとばしてもかまいません)。

以上のような方法によって音階表記で、8オクターブの音域をカバーすることができるのです。

それではもう1つの方法、つまりNnによる表記を説明します。これは、最低オクターブのCの音をN1、C#をN2、DをN3というように、音程が半音(100セント)上昇するごとに番号を1つ増やして、最高オクターブのBの音まで、Nの後ろに数字をかいて音の高さを指定する方法です。この方法も試してみましょう。

PLAY "N25N27N29N30N32N34N36N37"

どうですか？ ドレミファソラシドと演奏しましたね。この方法では、オクターブや半音の指定なども同様にできるので、場合によっては便利なこともあります。慣れないと使いにくいかもしれません。

これで音程は自由に設定できるようになりました。次は長さを決めてみましょう。

音の長さは基本的には音程を示すコード(Cなど)の後にその音の長さを示す1～64の整数を付け加えることによって設定できます。その数値はたとえば4分音符なら4、32音符なら

32というふうに長さの逆数表示ですから、従来の他の MML に比べて非常に使いやすくなっています。次の例を試してください。

PLAY"C4D8E16F32"

音の長さが変わったのがお分かりいただけたと思います。

もしここで音の長さの数値をつけないと、自動的にデフォルト値の長さに設定されます。

デフォルト値は電源投入時には4に設定されていますから、普通は4分音符になります。このデフォルト値も変更することができ、そのMMLはLnという形をとります。たとえばL8を指定すれば、とくに長さの指定のない音符はすべて8分音符になります。また、このデフォルト値もオクターブ指定と同様に、次に指定するまでは、その値が保持されます。次に例を示します。

PLAY"L8CDE16F16GAL2B"

音を休止したい場合には休符記号であるRを入れます。休符の長さは音の長さと同じようにR4で4分休符、R2で2分休符となります。ただし、後の数字を省略した場合は、Lで指定した長さにはならず、すべて4分休符となります。次の例を試してください。

PLAY"L2CR16CR8CR4CRC"

次に符点音符の出し方を説明します。符点音符を出す場合は実に簡単で、符点音符にしたい音のコードの後に"."(ピリオド)をつけるだけです。たとえばL8CDE.とすればEの音だけが符点8分音符、C2.とすればCの符点2分音符が出ます。

これで音の長さも自由に設定できるわけですが、音程をNnの形で設定した場合、音の高さを示す数字の後に続けて音の長さの数字をかくと、数字が3桁並んでMMLが判別できなくなるので、音程のコードの後に";"(セミコロン)を付けてそれに続けて音の長さのコードを書きます。たとえばN36;8という具合です。

それでは次に曲のテンポの設定をしましょう。実際の音楽では、同じ4分音符でも曲のテンポによってその長さが違います。N₆₀-BASICのMMLでは、テンポの設定のためにはTnという方法を使います。たとえばT90とすれば1/90、つまり1分間に4分音符が90回演奏できるテンポになります。ちなみに電源投入時にはT120に設定されています。またテンポの指定も一度指定すると新しく指定するまではその値が保持されます。

以上の方法を用いて、一声の音楽は一応演奏できるようになりました。それでは次に和声を出してみます。

PC-6001はGI社のAY-3-8910というサウンド用LSIを用いて音を発生させています。このLSIの特徴の1つに同時に3チャンネルの音を別個に設定できるということが挙げられます。N₆₀-BASICのMMLもこの特徴を活かして簡単に3つまでの和声のプログラミングが可能になっています。

例としてC、つまりドミソの和音をつくってみましょう。

PLAY"04C","04E","04G"

上の例でもわかるように、各チャンネルで演奏したい音を順に並べて書いて、その間を、(カンマ)で区切ると、その音を一齐に演奏します。これは別に3つでなくても2つだけ使うこともできます。

PLAY"C","G"

また、テンポ(T)、オクターブ(O)、長さ(L)等の指定は、各チャンネル別に行なうことができます。各自試してみてください。

これで3和声の音楽はプログラムできるようになったわけですが、これだけでは大変味気ない演奏になってしまいます。そこで、次に、音楽に変化を付ける方法を考えてみましょう。

音楽に変化を付けるための方法としては、1)音色を変える 2)音量、つまり音の大きさを
変える、等が考えられます。PC-6001の場合、音色に変化を付けることは、基本的には不可能ですから、ここでは音の大きさに変化を与える方法を説明します。

N₆₀-BASICのMMLには音量の変化を付ける方法として2種類の方法が準備されています。1つはVnによる方法、もう1つはSlとMnによる方法です。

まずVnによる方法とは、Vの後に0から15の整数を付けることによって、ボリューム、すなわち音量を変化させる方法です。V0を指定すると無音、それからV1、V2と数字を大きくするに依じて音も大きくなり、V15で最大になります。次の例を試してください。

PLAY"V1CV5EV9G"

音の大きさが変化しました。このVの指定もTやL等と同様に、1度指定すると次に指定するまで変わりませんし、チャンネルごとに別個に指定することが可能です。また電源投入時にはV8が設定されます。ちなみにVの数字と音声出力電圧の関係は指数的に変化します。これは、人間の耳の刺激に対する反応が対数的になっているのを補正するためだと思われます。

では次にSlとMnによる、エンベロープパターンの変化のさせ方について説明します。エンベロープパターン、つまり包絡線形状とは、シンセサイザに興味のある方は御存知でしょうが、簡単に言えば、音の大きさの時間に応じた変化の様子のことを言います。

たとえばギターやピアノは、音を発した瞬間は強い音がして、それから時間を経ると音が弱くなっていきます。またファゴットなどでは弱い音からふわっと立ち上がる感じの音がします。

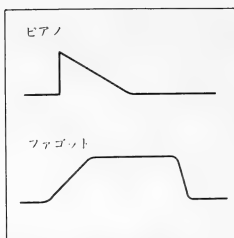


図 5-2 エンベロープパターン

この場合のエンベロープパターンを分かりやすく略して書いたものが図 5-2 です。もちろん実際はこんなに単純ではありません。

PC-6001 に使われているサウンド用 LSI は、このエンベロープをある程度自由に設定できます。このエンベロープ設定のための MML が S と M なのです。S はエンベロープの形状を、M はその周期を指定するものです。

S	エンベロープパターン
0, 1, 2, 3, 9	
4, 5, 6, 7, 15	
8	
10	
11	
12	
13	
14	

図 5-3

具体的には、エンベロープの形状は図5-3に示される8種類の中から選ぶようになっており、表に示された数字をSの後に付記して使います。数字は0～15の整数でなくてはなりません。周期は任意に設定可能で表5-3中のtの長さから公式で決定されます。その公式は

$$M = \frac{1996750 \times t}{256} \quad t: \text{周期(秒)}$$

という式です。

たとえば周期を2秒に設定したければ、

$$M = \frac{1996750 \times 2}{256} \approx 15600$$

ですからM15600を指定すれば良いわけです。この数字は1から65535までの自然数でなくてはなりません。

エンベロープを設定するとき、気を付けなくてはならないことが2つあります。第1に、2つ以上の異なったエンベロープを指定できないということです。つまり和声を出したときにチャンネルごとにSまたはMの値を変えることはできないのです。同じエンベロープを指定するか、他のチャンネルはエンベロープを指定せずにVで音量を指定するかしなければなりません。もう1つ気を付けるべきことは、同一チャンネルにエンベロープの指定、つまりSと、音量の指定、つまりVは同時にできないということです。たとえば、

PLAY"S0M1000V8....."

という指定をすると、後から指定したVの指定が有効になります。

以上で PLAY 命令の基礎の説明を終わります。これで一応の音楽演奏プログラムは作成可能になりました。例として簡単な音楽プログラムを紹介しておきます。

```

0 REM*****
1 REM* La fille aux cheveux de lin *
2 REM* *
3 REM* Composed by C. Debussy (Jan. 1910) *
4 REM* *
5 REM* Arranged & Encoded *
6 REM* by Yellow Panther *
7 REM* *
8 REM* To my dearest "Butch" *
9 REM*****
10 PLAY"T60L16S0M1000005","T60L16V10M1000003","
    T60L16V1003M10000"
20 PLAY"D-4. 0B-G-E-8G-B-05D-80B-G-E-8G-B-G-8G-E
    -","R1R4. B","R1R4. G-"
30 PLAY"G-8FE-M30000D-1M10000E-8G-8","03B-2","G
    -2"
40 PLAY"A-4. 05D-80B-8G-B-A-805D-80B-405","0F2G-
    4F2","D-2E-8D-4. D4"
50 PLAY"E-20B-4. B805D-4. 0B-G-E-8G-B-","B-2B-4. B
    80D-2E-4","E-2

```


60 PLAY"05D-80B-G-E-8G-B-G-8G-E-", "D-4E-4D-803B
 8", "03B-40C403B-8A-8"
 70 PLAY"G-8FE-D-1R803", "B-2. B-2", "G-40D-4D-03BB
 -A-G-8FE-D-4D-02B"
 80 PLAY"B-0D-E-8G-A-L8B05D-E-G-4L16", "R80D-2. B-
 4", "B-A-03B2. 0G-4"
 90 PLAY"FE-D-4V5D-S00BB-A-", "B-4", "G-403"
 100 PLAY"G-A-B+FE-A-G-D-03B0F", "E-8. FE-8. D-03BB"
 , "B8. 0D-03BB. B-A-8"
 110 PLAY"E-03B-A-0D-03C-8. 0D-F-A-B05D-F-A-", "BB-
 A-2. ", "A-G-F-2. "
 120 PLAY"BBB-A-G-2B-8A-G-E-4. D-0B", "05E-2. 0G-403
 B2", "0B1F2"
 130 PLAY"B-8A-G-E-4. E-C", "B4B-2", "G-402B-8R8B-4"
 140 PLAY"02B-8B-03CE-FGB-0C8E-C", "0E-203A-4", "03
 G2F4"
 150 PLAY"E-803B-0CE-FGB-05C8E-C", "02B-8. 03CE-FGB
 -0C8E-C", "G20F4"
 160 PLAY"E-80B-05CE-FG-B-06E-4. ", "G803B-0CE-FG-B
 -B4. ", "04B-205B4. "
 170 PLAY"L24D-E-D-05B-8A-8L16V5A-S0G-FE-L24", "05
 A-8G-8B4. 0", "F8E-20"
 180 PLAY"E-8D-E-D-L160B-8A-4G-A-", "B8A-8G-8A-403
 G-0F", "G-8F8E-8C4B8"
 190 PLAY"L8B-. G-16E-G-B-05", "L8E-. D-1603B-0D-E-"
 , "L803B-. G-16E-G-B-0"
 200 PLAY"D-0B-G-E-4G-4B-. ", "G-E-D-03B-4B40E-. ", "
 D-03B-G-E-4G-4B-. "
 210 PLAY"G-16B-05D-E-G-B-G-", "D-16E-G-B-05D-E-D-
 ", "G-16B-0D-E-G-B-G-"
 220 PLAY"0B-4D-4R206D-4V4D-4. S0L16", "03C-4A-40G-
 1G-", "03G-4F40E-1E-"
 230 PLAY"05B-G-E-8G-B-06D-805B-G-E-8G-B-G-8G-E-"
 , "G-G-1", "E-E-1"
 240 PLAY"LG-FE-D-. 0L8BB-A-L16", "B-2. D-2. L16", "G-
 1. L1603"
 250 PLAY"G-A-B-FE-A-G-D-03B0F", "E-8. FE-8. D-03BB"
 , "B8. 0D-03BB. B-A-8"
 260 PLAY"E-03B-A-0D-E-03B-B0", "BB-A-8. B-B", "A-G-
 FB. G-A-"
 270 PLAY"D-E-03B-A-0", "BBB-A-", "A-8G-F-"
 280 PLAY"D-E-03B-B0G-A-B05D-E-G-A-", "A-80E-8D-E-
 G-A-B05D-E-", "F8G-A-2"
 290 PLAY"L12B06D-E-. ", "L12G-A-BB", "A-4A-24"
 300 PLAY"S0M600000G-1. ", "S006R16D-1. ", "S005R32B-
 1. "

5-2 SOUND 命令

これまでPLAY命令を使った簡単なプログラミングテクニックを紹介してきました。しかしPLAY命令では音の高さの変化をなめらかにしたり、ノイズを発生したりすることはできません。このような効果音を発生するためにはサウンド用LSIを直接コントロールする必要があります。サウンド用LSIをBASICで直接コントロールするための命令が、これから説明するSOUND命令です。

SOUND命令は次のような形になります。

SOUND [レジスタ番号], [データ]

レジスタとは、一種のメモリのようなもので、それぞれに音の高さとか大きさといったデータを入れます。それではこのレジスタについて少し詳しく説明します。

PC-6001のサウンド用LSIのレジスタは付録-10のようになっています。

まず[R0, R1], [R2, R3], [R4, R5]の3つのレジスタペアは、それぞれA, B, C各チャネルの音の周波数を決定します。R0, R2, R4がそれぞれの微調整、R1, R3, R5が主調整で、前者は0～255の整数、後者は0～15の整数がデータとして有効になります。そして出力したい周波数から公式を使って、このデータを求めることができます。その公式は次のとおりです。

$$TP = \frac{1996750}{16 \times f_T}$$

$$TP = CT \times 256 + FT$$

f_T : 出力する周波数 (Hz)

CT : 主調整レジスタの値(R1, 3, 5)

FT : 微調整レジスタの値(R0, 2, 4)

たとえばAの音、440Hzの音を出すときには、

$$TP = \frac{1996750}{16 \times 440} \approx 284$$

$$284 = 1 \times 256 + 28$$

ですからCT=1, FT=28となります。したがって、R0には28, R1には1を設定すればよいわけです。

次にR6は、ノイズの周波数を設定します。このLSIはノイズ発生器を1つ内蔵しており、その周波数を設定できるのです。このレジスタは0～31が有効で、次の公式によってその値を決定します。

$$NP = \frac{1996750}{16 \times f_N}$$

f_N : ノイズ周波数 (Hz)

NP : [R6] の値

次に R7 です。このレジスタは、ミクサーの役割りをします。このレジスタの各ビットが、それぞれのチャンネルのトーンとノイズ、そしてこの LSI を汎用 I/O ポートとして使う場合の入出力の設定を意味します。ただしこれは負論理です。もっとも、この説明では何のことやら分からない人もいられるでしょうから、そういう人のために、このチャンネルに設定するデータと、各チャンネルのトーン、ノイズの出力の様子を表にしたものを示します(図 5-4)ので、参考にして下さい。

表中 &H…と書かれたものが16進表示の R7 のデータ、その右が出力されるトーンとノイズ、およびそのチャンネルです。たとえば R7 に F1H を設定すると A チャンネルはノイズのみ、B、C チャンネルはトーンのみが出力されます。

R8, R9, R10 はそれぞれチャンネル A, B, C の音量設定、PLAY 命令の V にあたるものです。このレジスタは 0 ~ 15 を設定すると、それに応じた音量になりますが、16 を設定すると、そのチャンネルの音量はエンベロープジェネレータによって支配されます。つまり、そのチャンネルはエンベロープのかかった音になるわけです。

R11, R12 は、エンベロープジェネレータの周期を決定するレジスタペアで、この値は PLAY 命令の M にあたるものです。各レジスタとも 0 ~ 255 が有効で、周期からデータを算出します。その値は PLAY の M の式で求めた値から、

$$EP = CT \times 256 + FT$$

EP : M の値

CT : 主調整レジスタの値 [R12]

FT : 微調整レジスタの値 [R11]

という式を使って求めます。

R13 はエンベロープの形状を決定するレジスタで、PLAY 命令の S に相当するもので、このレジスタに値を与えた瞬間からエンベロープの発生が始まります。

ですから、R13 以外のレジスタの値をくずさなければ、後は SOUND13, [データ] を実行すれば何度でも同じ音が出るわけです。

また、このデータを 8 や 10, 11, 12, 13, 14 にすると、後は何もしなくても勝手に音が連続して発生されます。その間 CPU は自由に仕事ができますから、これを上手に利用すれば、ゲームのバックグラウンドに音を出すなどということは大変簡単にできるわけです。

R14, R15 は I/O ポートのデータのレジスタですから、音の発生には直接関係ありません。

以上の方法で、各レジスタのデータが決定したら、これらを実際にレジスタにセットすれ

ば音が出るわけです。

たとえばチャンネルAの音量を最大にしたければ、

SOUND 8,15

また、チャンネルBにエンベロープを付けたければ、

SOUND 9,16

&HFF	出力なし	&HDF	Cノイズのみ
FE	Aトーンのみ	DE	CノイズAトーン
FD	Bトーン	DD	Bトーン
FC	ABトーン	DC	ABトーン
FB	Cトーン	DB	Cトーン
FA	ACトーン	DA	ACトーン
F9	BCトーン	D9	BCトーン
F8	ABCトーン	D8	ABCトーン
F7	Aノイズのみ	D7	ACノイズのみ
F6	AノイズAトーン	D6	ACノイズAトーン
F5	Bトーン	D5	Bトーン
F4	ABトーン	D4	ABトーン
F3	Cトーン	D3	Cトーン
F2	ACトーン	D2	ACトーン
F1	BCトーン	D1	BCトーン
F0	ABCトーン	D0	ABCトーン
EF	Bノイズのみ	CF	BCノイズのみ
EE	BノイズAトーン	CE	BCノイズAトーン
ED	Bトーン	CD	Bトーン
EC	ABトーン	CC	ABトーン
EB	Cトーン	CB	Cトーン
EA	ACトーン	CA	ACトーン
E9	BCトーン	C9	BCトーン
E8	ABCトーン	C8	ABCトーン
E7	ABノイズのみ	C7	ABCノイズのみ
E6	ABノイズAトーン	C6	ABCノイズAトーン
E5	Bトーン	C5	Bトーン
E4	ABトーン	C4	ABトーン
E3	Cトーン	C3	Cトーン
E2	ACトーン	C2	ACトーン
E1	BCトーン	C1	BCトーン
E0	ABCトーン	C0	ABCトーン

図5-4

を入力します。

SOUND 命令について理解されたでしょうか？ なんだか、ミュージック・シンセサイザみたいですね、参考のためにこのサウンド用 LSI のブロックダイアグラムを図 5-5 に示します。

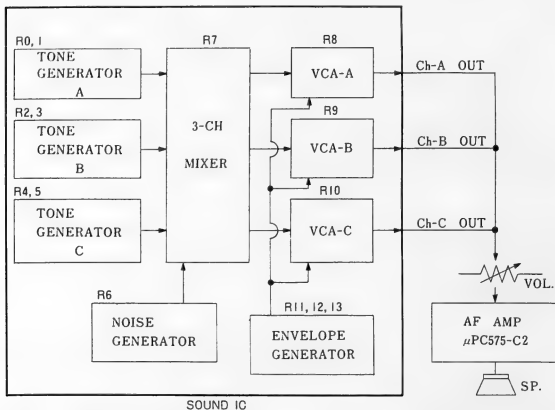


図 5-5 AY-3-8910 ブロックダイアグラム

それでは次に、サウンド命令を使ったいろいろな効果音の例を示しておきます。自作のゲーム等に応用してください。

ロケット発射音のような音

```

10 SOUND1,0: SOUND3,0: SOUND12,255: SOUND11,255: SOUND6,31: SOUND7,244
15 SOUND8,15: SOUND9,0
20 FOR I=200TO140STEP-0.2: SOUND0,I: NEXT: SOUND9,15
25 FOR I=140TO40STEP-0.2: SOUND0,I: SOUND2,I+60: NEXT: SOUND8,16: SOUND9,16
30 SOUND13,0: FOR I=40TO10STEP-0.2: SOUND0,I: SOUND2,I+60: SOUND6,I-9: NEXT
  
```

ヘリコプターのような音

```
10 SOUND0,20: SOUND1,0: SOUND2,30: SOUND3,0: SOUND4
   ,0: SOUND5,9: SOUND6,0
20 SOUND7,48: SOUND8,16: SOUND9,4: SOUND10,6: SOUND
   11,100
30 SOUND12,2: SOUND13,12
```

爆発音1

```
10 SOUND6,31
20 SOUND7,246
30 SOUND8,255: SOUND1,15
40 SOUND8,16
50 SOUND11,255
60 SOUND12,255
70 SOUND13,0
80 FOR T=1TO250:NEXT: SOUND13,0
```

爆発音2

```
10 SOUND6,5
20 SOUND7,247
30 SOUND8,16
40 SOUND11,0
50 SOUND12,20
60 SOUND13,0
```

タイプライターのような音

```
10 SOUND0,20
20 SOUND1,0
30 SOUND2,20
40 SOUND3,20
50 SOUND4,10
60 SOUND5,0
70 SOUND6,8
80 SOUND7,224
90 SOUND8,16
100 SOUND9,16
110 SOUND10,16
120 SOUND11,0
130 SOUND12,5
140 SOUND13,0
150 REM コナト SOUND 13,0 ヲ ショコウスレバ ソノタヒニ オトカ
   テマス。
```

5-3 より高度なテクニック

5-3-1 PLAY のバッファ

今まで PLAY 命令を使ってきて、観察力のある方は気付かれたと思いますが、PLAY 命令を実行すると、音が全部演奏される前に、すでに PC-6001 は Ok を表示して、カーソルが点滅し、入力待ち状態になっています(気付いていなかった人は今すぐ試してください)。

これはどういうことかといいますと、PC-6001 は内部に PLAY 命令専用のバッファをもっており、PLAY 命令が与えられると、すべてのデータをバッファに取り込み、あとはタイマ割り込みで取り込んだデータを演奏するしくみになっているのです。

つまり、すべてのデータをバッファに取り込んだ時点で、PLAY 命令の実行は終了したと判断し、次に命令があれば、バッファ内のデータを演奏しながら、その命令の実行に移るのです。ですから PLAY 命令を 1 つ与えると、音を演奏している途中ですでに次の命令の待ち受け状態になっているわけです。

このことは上手に使えば、タイマ割り込みを使った音の出力と、他の仕事の同時進行が、BASICのみで可能になります。

次のプログラムは PC-6001 用マージャンゲームのキー入力ルーチンの例で、“ピッポッ”という音を連続出力しながら捨てハイの入力を INKEY\$ で受けます。

```
610 FOR I=1 TO 68
620 AS=INKEY$: IF AS="" THEN NEXT I:PLAY "l16v8o3g
-r8. e-r8. ":GOTO 610
```

ところで、逆にこれが非常にじゃまになることも当然あります。

たとえば音楽の演奏の終了と同時に画面を書き換えたい等という要求があった場合、PLAY 命令の後に続けて画面書き換えの命令を書くとも演奏が終了しないうちに画面が書き換えられてしまいます。ですから何らかの方法で演奏終了まで画面操作を待ってやらなければなりません。

まず考えられるのは FOR-NEXT 等を使って時間待ちをする方法です。

しかしこの方法では待ち時間の調整が大変微妙で、音楽演奏の終了と同時に次の命令の実行に移らせるのはむずかしいでしょう。

ではどうすればよいでしょうか。

実は、N₆₀-BASIC のワークエリアには、PLAY 命令中での演奏中のチャンネル数、つまり残りのチャンネル数を示している部分があって、FD1BH 番地の下位からの 3 ビットがそれぞれのチャンネルに対応して、正論理で演奏中か否かを表わしています。演奏が完全に終了すれば、このアドレスのデータは 0 になりますからこれを利用すれば演奏終了と同時に次の仕事に移ることができるのです。

次のプログラム例を見てください。音楽が終了すると同時に“END”と画面に表示します。

```

10 CLS
20 PLAY "L2C","L2E","L2G"
30 IF PEEK(&HFD1B)<>0 THEN 30
40 PRINT "END"

```

5-3-2 PLAY に変数を

次に PLAY 命令中に変数を使う方法を説明しましょう。

PLAY に変数を使うといえば、まず頭に浮かぶのは文字数に PLAY のデータを代入して使う方法でしょう。たとえば、

```

10 A$="CDE":B$="FGA"
20 PLAY A$+B$

```

といった具合です。

しかし、ここでいう変数は文字変数ではありません。ごく普通の数値変数です。PLAY 命令中で使用する数値と言えば、O や V, L, N などを与える数値ですが、これに変数を使うわけです。

PLAY 命令中に変数を使う場合は次の方法で行ないます。

```
PLAY "[MML]=[変数名];"
```

具体的には、

```

10 FOR NUMERIC=1 TO 96
20 PLAY "N=NUMERIC; 32R32"
30 NEXT NUMERIC

```

となります。ここでは NUMERIC というのが変数名で、この変数を FOR~NEXT ループを使って 1 から 96 まで変えて、これを MML の中の "N" に付けて演奏します。もう 1 つの例を示しましょう。

```

10 OCTAVE=INT(8*RND(1))+1
20 PLAY "O=OC; C", "O=OC; E", "O=OC; G"
30 GOTO 10

```

この例ではドミソの和音の高さを乱数で決定しています。10 行の OCTAVE が、オクターブを代入する変数です。20 行では OC という変数名になっていますが、変数は最初 2 文字のみ有効ですからこれでよいのです。

それからこれは余談ですが、V や L, O, T は、その後の数値を省略してもかまいません。

ただし、省略した場合は、自動的にデフォルト値、つまり電源投入時の値に設定されます。すなわち、

PLAY "TLVOCEG" と、

PLAY "T120L4V804CEG"

は、全く同じ働きなのです。ですからメモリ節約のためにも、この省略は上手に応用しましょう。

5-3-3 PLAY と音色

PC-6001 に使われているサウンド用 LSI のトーン出力は、すべて方形波です。つまり、1つの音色しか出せません。これではどうしても単調な感じになってしまいます。それを少しでも解決する方法を考えてみましょう。

まず、パーカッションの音を PLAY 命令で出してみましょう。

まず、ハイハットの音です。これは高めのノイズにエンベロープを付ければよさそうです。しかし、PLAY 命令ではノイズを出すことはできません。そこで PLAY と SOUND を組み合わせさせてみましょう。

```
10 SOUND 6,0
20 SOUND 7,&HF1
30 PLAY "S0M3000C"
```

上のプログラムを試して下さい。なんとなくそれらしい音が出ました。10行はノイズの音の高さの設定、20行はAのチャンネルにノイズだけを、B、Cのチャンネルにトーンだけを出力するように設定しています。あとは PLAY 命令で、好きなようにノイズが演奏できるわけです。30行の最後のCは、別にCでなくてもA～Gであれば何でもかまいません。

次に、ノイズの音を少し変えて、低いトーンを混ぜて、スネアドラムのような音を出してみよう(そう聞こえるかどうかはあなたの感性におまかせしますが…)。

```
10 SOUND 6,10
20 SOUND 7,&HF0
30 PLAY "S0M3000L1603E8EEEEEE8"
```

今度はチャンネルAはトーンも出力されていますから30行のEを変えると当然音が変化します。それから上の2つの例では20行を変えればノイズの出るチャンネルが変わるのがお分かりいただけると思います。たとえば、ハイハットの例で、

20 SOUND 7,&HD8

とすれば、30行は PLAY [A], [B], "S0M3000……" となって、A、Bの部分でメロディーを演奏して、チャンネルCはハイハット、となるわけです。

では、次はノイズを全く使わずにつくるパーカッションの音です。次のプログラムを試してください。

```
10 PLAY "S0M10003T150L16C8C+C+D8D+D+E8FFFF+8GGG+8  
GGF+8FFE+8EED+8DD"  
20 GOTO 10
```

いかがですか？ まだまだできそうですね。

パーカッションはこれでよいとして、メロディーの部分はどうしたらよいのでしょうか。

PLAY 命令で音色に変化をもたせる簡単な方法は、複数のチャンネルを使って1つのメロディーを演奏させることです。たとえば、

```
PLAY "S13M100TLOCEG"
```

と、

```
PLAY "S13M100TLOCEG", "S13M100TLO5CEG"
```

とでは大変違った感じの音になります。

チャンネルBにチャンネルAの2倍の高さ、つまり1オクターブ上の音を重ねてみたのです。

この方法は、チャンネル数を1チャンネル多く使うという欠点がありますが、大変簡単であり、またチャンネルごとに音量を変えたり、混ぜる音の高さの比を変えたりすることによって豊富な変化が得られ、効果的な方法です。各自試して、自分の耳で確かめてください。

さて、この場合に限らず、エンベロープを使うチャンネルと、使わないチャンネルとで、和声が発生する場合、気を付けねばならないことがあります。これは PLAY 命令の BUG の1つですが、状況によって、エンベロープの周期がくずれてしまうのです。くずれるというのは具体的に言えば周期が勝手に書き換えられてしまうのです。

1つ試してみましょう。

```
10 PLAY "T05S0M3000L8AA16G16FAG406C4", "TULFDEG"  
20 PLAY "05EF16E16DDCE16G1606C4", "EG8E8C4"
```

いかがですか？ S0M3000 には極端にエンベロープの周期が短いんですね。PLAY 命令をちょっといじったことのある方ならこのような症状に出合ったことがあるでしょう。これを正常にするには、次のようにエンベロープに関係のないチャンネルにもエンベロープ周期(M)の指定をします。

```
10 PLAY "T05S0M3000L8AA16G16FAG406C4", "TUM3000L  
FDEG"  
20 PLAY "05EF16E16DDCE16G1606C4", "EG8E8C4"
```

今度はまともな音が出ましたね、覚えておくと便利でしょう。

5-3-4 サウンド機能と機械語

PC-6001 のサウンド機能は当然機械語で制御できます。その方法について説明しましょう。
サウンド用 LSI とのコミュニケーションのための I/O ポートは、A0～A3H です。(付録-1 参照)

このうち A0H がサウンドレジスタのアドレスラッチです。ここにサウンドレジスタの数値を出力することにより、そのレジスタとのコミュニケーションが可能になります。

A1H にデータを出力すると、そのデータが A0H により指定したレジスタに入力され、また A2H には、A0H によって指定したレジスタに入っているデータが現われます。

試しに BASIC の OUT, INP 命令で直接 I/O ポートをアクセスして、SOUND 命令と比べてみましょう。まず、

PLAY "a"

を実行して下さい。その後で、

SOUND 8,8

を実行すると、a の音が連続して出ますね。

その後、

SOUND 8,0

を実行すれば音が止まります。次に続けて I/O ポートに直接 OUT 命令で出力してみましょう。

OUT&HA0,8 : OUT&HA1,8

これは SOUND 8,8と同じです。

OUT&HA0,8 : OUT&HA1,0

これは SOUND 8,0と同じです。

今度はレジスタのデータを読んでみましょう。

OUT&HA0,0 : ? INP(&HA2)

28

サウンドレジスタの0番には、28というデータが入っていることが分かりました。

これらはもちろん機械語でも実行できます。

以上のように直接I/Oポートをアクセスすれば、サウンドレジスタにデータが書けることが分かりました。しかしこれはあまりスマートな方法ではありません。そこで、次に N₆₀-BASIC の内部ルーチンを使って、サウンドレジスタを制御してみましょう。

N₆₀-BASIC のサウンドレジスタコントロール用の内部ルーチンは 1BC5H から始まっています。Aレジスタにレジスタ番号、Eレジスタにデータをセットして、このルーチンを CALL すると SOUND 命令と同じ効果が得られます。

たとえば SOUND 7,&HF0 でしたら、

```
0000 3E07      SOUND: LD   A,7
0002 1EF0              LD   E,0F0H
0004 CDC51B          CALL 1BC5H
0007 C9              RET
```

でよいわけです。

ゲームの効果音等、変化の激しい音は BASIC では難しいのですが、このような方法を使って機械語で出せば大変簡単です。

さて、ゲーム等、機械語でつくったプログラム中で困るのは音楽の演奏です。効果音と違って、音楽は音の長さ等を正確に設定してやらねばなりません。そこで、音楽を機械語で演奏するときは PLAY 命令の内部ルーチンを使うのがよいでしょう。

PLAY 命令の内部ルーチンの CALL 番地は 1EB3H です。HL レジスタに、演奏したい音楽を、MML で表記したデータ (BASIC の PLAY 命令のときと同じ形式) が格納されているメモリの先頭アドレスを入れて、このルーチンを CALL すれば PLAY 命令と同じ効果が得られます。

次に簡単な例を示します。

```
                                ORG  0D000H
D000 2109D0  PLAY: LD   HL,DATA
D003 7E              LD   A,(HL)
D004 B7              OR   A
D005 CDB31E          CALL 1EB3H
D008 C9              RET

                                ;
D009 2243222C DATA: DB   "C","E","G",0
D00D 2245222C
D011 22472200
```

注) データの終りを示すエンドマーク
の00を忘れないようにすること

前にも述べましたとおり、機械語を使ってサウンド用 LSI をコントロールすると、非常に変化の速い複雑な波形をもった音も、簡単に出すことが可能です。

極端な例としては、任意の波形を持った音(たとえば人間の音声)をつくり出すことも可能です。

PC-6001 に使われているサウンド用 LSI は、周波数のレジスタ(たとえばレジスタ 0 とレジスタ 1)に 0 を入れると、出力には音量レジスタ(レジスタ 8~10)で指定した直流電圧が発生します(この LSI のマニュアルにはそう発表されてはいないらしいのですが、実際は、こう考えて問題ありません)。ですから、周波数レジスタに 0 を入れておき、音量レジスタの値をすばやく変化させてやれば、任意の波形が出力されるわけです。

最もこれは BASIC では当然速度が不足しますので、機械語でなくてはまともな音は出ません。また音量レジスタの値と実際に出力される電圧は指数関係にあるので、注意してください。

それでは最後にサウンド開発のツールとなるプログラムを公開します。

SOUND 命令で音を上手につくるためのヒケツは、やはり習うより慣れろ、理論よりもまず実践です。自分で PC-6001 を使いながら体得するほかありません。その場合に役に立つのがこのプログラムです。

すべて BASIC で書かれていて、RUN させると画面右側に、reg. NO. ? と聞いてきますのでコントロールしたいレジスタの番号を入力してください。

するとさらに、data? と聞いてきますのでそのレジスタに入りたいデータを入力してください。画面の左側にはその時点での全サウンドレジスタの内容が表示されます。

なお、レジスタ番号、データはすべて10進数です。入力時に16進数を使いたいときには &HXX という形式で入力してください。

F・1 のキーに &H が入っています。

```
1 REM*****
2 REM*   Sound Creating Tool   *
3 REM*           for PC-6001   *
4 REM*   Copyright (C) 1981   *
5 REM*       Yellow Panther   *
6 REM*****
10 SCREEN 1,1,1:CONSOLE 0,16,0,1:COLOR 0,0,1:CLS:KEY1,"&H"
15 FOR N=0 TO 13:GOSUB 200:NEXT
20 LOCATE 0,0:PRINT "Reg. —Data":PRINT "———"
30 LOCATE 18,18:PRINT "          ":LOCATE 18,18:INPUT"reg.No.":N
40 IF N>13 OR N<0 THEN 30
50 LOCATE 18,18:PRINT "          ":LOCATE 18,18:INPUT" data":D
60 IF D>255OR N<0 THEN 50
70 SOUND N,D
100 GOSUB 200
120 GOTO 30
200 LOCATE 0,N+2:OUT &HA0,N:A2=INP(&HA2)
210 PRINT TAB(1+(N>9))N;TAB(8+(A2>9)+(A2>99))A2;
220 RETURN
```


第6章 カセット

- 6-1 ボーレート
- 6-2 フォーマット
 - 6-2-1 プログラムファイル
 - 6-2-2 データ・ファイル
- 6-3 PC-8001のデータをPC-6001で使用する
- 6-4 PC-8001のプログラムテープをPC-6001で
LOADする
- 6-5 BASICと機械語を一度にSAVE・LOADする
- 6-6 データ・ファイルにおける, と;の違い

第6章 カセット

6-1 ボーレート

PC-6001 では、カセットのボーレートを600または1200ボーに切り換えることができます。
電源 ON 時では1200ボーにセットされています。

FA1FH 番地がボーレートの切り換えのフラグになっており、電源 ON のときの FA1FH 番地を調べてみましょう。

```
How Many Pages? 2
N60-BASIC
By Microsoft (c) 1981
23484 Bytes free
Ok
?peek(&hfa1f)
255
Ok
```

FA1FH 番地の値が0の時に600ボー、0以外のときは1200ボーになります。

初期に出荷された製品の取り扱い説明書では、この部分が間違って書かれていますので注意が必要です。

ハード的には、PC-8001、および PC-8801 のカセットを PC-6001 で読み取ることが可能です。

6-2 フォーマット


6-2-1 プログラムファイル

PC-6001では、BASIC のプログラムファイルが、実際どのような型式で SAVE されるか調べてみましょう。

```
100 POKE&HFFE6,0:POKE &HFFE7,&HE
2
110 FOR I=&HFFE8 TO &HFFF8:READ
A$:POKE I,VAL("&h"+A$):NEXT
120 POKE &HFA00,&HE8:POKE &HFA09
,&HFF
200 DATA F5,CD,78,0E,32,1D,FA,E5
,2A,E6,FF,77,23,22,E6,FF,E1,C3,A
6,0F
Ok
RUN
Ok
csave"test"
```

上記のプログラムを実行して、カセットに SAVE します。その後、テープを巻き戻して、今、SAVE したプログラムを LOAD します。

```
cload
Found:test
Ok
```

LOAD 終了後  キーを押してページ 2 に切り換えます。

```
EEEEEEEEtest 太Od 2&HFFE6,0:2
&HFFE7,&HE2 Kon ♥ Ix&HFFE8 テ &HF
FFB:を A$:2 I,め("&h")A$):2 kOx 2
&HFA00,&HE8:2 &HFA09,&HFF 102 ♥
F5,CD,78,0E,32,1D,FA,E5,2A,E6,FF
,77,23,22,E6,FF,E1,C3,A6,0F
```

ページ 2 に表示されている内容が、テープに記録されていた内容になります。

	ヘッダ										ファイル名			
E200	D3	D3	D3	D3	D3	D3	D3	D3	D3	D3	74	65	73	74 00 00
E210	1D	84	64	00	94	26	48	46	46	45	36	2C	30	3A 94 20
E220	26	48	46	46	45	37	2C	26	48	45	32	00	4B	84 6E 00
E230	B1	20	49	D2	26	48	46	46	45	38	20	C3	20	26 48 46

E240	46	46	42	3A	86	20	41	24	3A	94	20	49	2C	E7	28	22
E250	26	68	22	CA	41	24	29	3A	82	00	6B	84	7B	00	94	20
E260	26	48	46	41	30	38	2C	26	48	45	38	3A	94	20	26	48
E270	46	41	30	39	2C	26	48	46	46	00	AD	84	C8	00	83	20
E280	46	35	2C	43	44	2C	37	38	2C	30	45	2C	33	32	2C	31
E290	44	2C	46	41	2C	45	35	2C	32	41	2C	45	36	2C	46	46
E2A0	2C	37	37	2C	32	33	2C	32	32	2C	45	36	2C	46	46	2C
E2B0	45	31	2C	43	33	2C	41	36	2C	30	46	00	00	00	00	00
E2C0	00	00	00	00	00	20	20	20	20	20	20	20	20	20	20	20

エンドマーク

プログラム

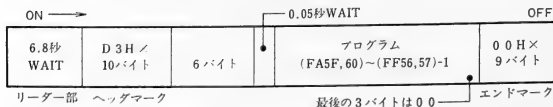


図 6-1 プログラムファイルのフォーマット

図 6-1 を見てください。まずモーターを ON にして 3.4 秒間テープをから送りします。(これはモーターの回転を安定させるために必要です)

次に同じく 3.4 秒間、マーク (2400Hz 4 サイクル) を出力します。そしてヘッダ・マークである D3H を 10 個書き込みます。LOAD するときに D3H を見て、プログラム・ファイルであると判断されます。

次にファイル名を 6 バイト書くわけですが、もしファイル名が 6 文字に満たないときは、後に 00 を書き込みます。

続いて 0.05 秒間テープを進めます。これはこのときに、LOAD 時のファイル名チェックや、Found: ..., Skip 表示などの処理を行なわせるためです。

この後よりプログラムになります。(FA5F, 60H) で指されるプログラムの先頭番地から (FF56, 57H の変数開始番地) - 1 までをカセットに出力します。これはメモリに入っている内容がそのまま送られます。

最後にエンドマークの 00 を 9 つ書き込みます。プログラムの最後に 00 を 3 つ送っていますので 00 は合計 12 個書き込まれることになります。

LOAD のときに 00 が 10 個連続していればファイルエンドとみなされます。

データをすべて転送した後はモーターを OFF にして SAVE を終わります。

6-2-2 データ・ファイル

データ・ファイルの構造は次のようになっています。

RUN	
1	AB
2	BC
3	CD
4. 5	DE
5. 3	EF
6. 4	FG
7. 7	GH
8	HI
9	IJ
10	JK
Ok	

1 COLOR CLOAD GOTO LIST RUN

PC-6001で上記のプログラムを実行すると、PC-8001 で SAVE されたデータ・ファイルが正確に読み取られています。

同様に PC-8801 のデータ・テープも読み取ることができます。

```

100 CLEAR 1000: DIM DA(10), DA$(10)
110 FOR I=1 TO 10: READ DA(I): NEXT I
120 FOR I=1 TO 10: READ DA$(I): NEXT I
130 OPEN "CAS1:data" FOR OUTPUT AS #1
140 FOR I=1 TO 10: PRINT #1, DA(I), DA$(I): NEXT I
200 DATA 1,2,3,4.5,5.3,6.4,7.7,8,9,10
210 DATA AB,BC,CD,DE,EF,FG,GH,HI,IJ,JK

```

※PC-8801 のプログラム

```

100 CLEAR 1000: DIM DA(10), DA$(10)
110 POKE &HFA1F, 1—————1200ボーにセット
140 FOR I=1 TO 10: INPUT #1, DA(I), DA$(I): NEXT I
150 FOR I=1 TO 10: PRINT DA(I), DA$(I): NEXT I

```

6-4 PC-8001のプログラムテープをPC-6001でLOADする

PC-8001 のデータ・テープを PC-6001 で読むことができるならば、プログラムテープもハード的には読むことができるはずですが、

先ほどのプログラムを SAVE して PC-6001 で LOAD してみましょう。

```
100 CLEAR 1000: DIM DA(10), DA$(10)
110 FOR I=1 TO 10: READ DA(I): NEXT I
120 FOR I=1 TO 10: READ DA$(I): NEXT I
140 FOR I=1 TO 10: PRINT #1, DA(I), DA$(I): NEXT I
200 DATA 1,2,3,4,5,5,3,6,4,7,7,8,9,10
210 DATA AB,BC,CD,DE,EF,FG,GH,HI,IJ,JK
```



※PC-8001のプログラム

LIST

```
100 LPRINT ASC: READ DA(
), DA$(
)
110 NEXT I TIME USR : LET DA(I): DA
TA I
120 NEXT I TIME USR : LET DA$(I): D
ATA I
140 NEXT I TIME USR : ON #47: 8 , DA
(I), DA$(I): DATA I
200 INPUT 1,2,3,4,5,5,3,6,4,7,7,
8,9,10
210 INPUT AB,BC,CD,DE,EF,FG,GH,H
I,IJ,JK
Ok
```

※PC-6001でLOADし、LISTしたときのプログラム、
PC-6001のボーレートは600ボーにすること

PC-8001 と PC-6001 では、中間言語や数値型の格納の仕方の違いから LIST を取っても正常なプログラムになっていません。

プログラムを中間言語レベルではなく、文字列の型で SAVE を行なえば、少なくとも PC-6001 で命令が変わることはありません。

プログラムを文字列で SAVE する一番簡単な方法は、プログラムの先頭に、" をつけければよいのです。

こうすると中間言語交換のときに、" があるために文字列と解釈されて中間言語に変換されません。

```

100 "CLEAR 1000: DIM DA(10), DA$(10)
110 "FOR I=1 TO 10: READ DA(I): NEXT I
120 "FOR I=1 TO 10: READ DA$(I): NEXT I
140 "FOR I=1 TO 10: PRINT #-1, DA(I), DA$(I): NEXT I
200 "DATA 1, 2, 3, 4, 5, 5, 3, 6, 4, 7, 7, 8, 9, 10
210 "DATA AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK

```

↓
 各ステートメントの先頭に " をつけて、PC-8001で SAVE
 します、そして PC-6001で LOAD します

```

100 "CLEAR 1000: DIM DA(10), DA$(10)
110 "FOR I=1 TO 10: READ DA(I): NEXT I
120 "FOR I=1 TO 10: READ DA$(I): NEXT I
140 "FOR I=1 TO 10: PRINT #-1, DA(I), DA$(I): NEXT I
200 "DATA 1, 2, 3, 4, 5, 5, 3, 6, 4, 7, 7, 8, 9, 10
210 "DATA AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK

```

↓
 " を取り除きます

```

100 CLEAR 1000: DIM DA(10), DA$(10)
110 FOR I=1 TO 10: READ DA(I): NEXT I
120 FOR I=1 TO 10: READ DA$(I): NEXT I
140 FOR I=1 TO 10: PRINT #-1, DA(I), DA$(I): NEXT I
200 DATA 1, 2, 3, 4, 5, 5, 3, 6, 4, 7, 7, 8, 9, 10
210 DATA AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK

```

この程度のプログラムでしたら PC-6001 でも実行できますが、PC-6001 には命令を使用していた場合はプログラムの修正が必要になります。

6-5 BASICと機械語を一度にSAVE・LOADする

SAVE のときに、FA5F, 60H で指される番地から、FF56, 57Hで指されるメモリのアドレス-1までテープに書き込まれます。そしてLOADするときは00が10個でくるまで読みます。このことはSAVEするときにFF56, 57Hの番地の値を変更すればBASICとその後に続く機械語のプログラムをSAVEできることになります。

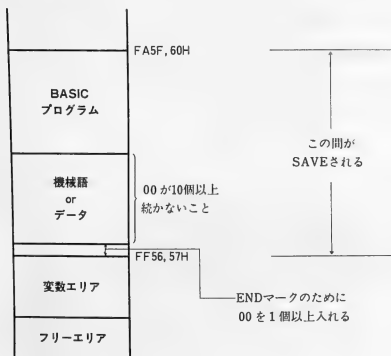


図6-3

これを行なう方法は次のとおりです。

1. BASIC のプログラムを入力する。
2. FF56, 57H の値を書き換えて機械語の入るエリアを確保する。
3. 機械語のプログラムを入力する。(機械語のプログラムの最後には必ず00が1個以上あること)
4. CSAVE を行なう。

ただし、注意することが3つあります。

1. FF56, 57H のポイントを書き換えたあとはBASIC のプログラムを修正してはいけません。もし修正を行なうと、プログラムが増減した分だけ機械語の位置がずれます。
2. 機械語のプログラムの途中で00が10個以上あってはいけません。
3. 機械語のプログラムをリロケータブルにつくっていない限り、SAVEしたときのRAM容量と同じでないとLOADしたときに正常に動きません。

市販のソフトには、このように BASIC+機械語の形で SAVE されているプログラムを多数みかけます。

6-6 テータファイルにおける、と；との違い

複数のデータを1度にカセットにSAVEしようとするとき、PC-8001では、PRINT#-1, A, B, C, Dのように行なえばよかったのですが、PC-6001では、PRINT#-1, A, ", ", B, ", ", C, ", ", Dのようにしなせんと INPUT#-1で読み込んだときにFDエラーになります。PC-8001の場合はデータの区切りとして","を自動的に送り出すのに対して、PC-6001は送り出さないために変数AとBとがつながって文字列とみなされるためにFD Error(File Data Error)となるのです。

そのためPC-6001はセパレータとしてプログラムで","を書く必要があります。

ところでPC-6001ではPRINT#-1, A, ", ", BとするよりもPRINT#-1, A; ", "; Bとする方が、データのSAVEの時間が短くてすみます。これはなぜでしょうか？

まず、','を使った場合を調べてみましょう。

```
10 FOR I=1 TO 5:READ A$:A=VAL("&H"+A$)
20 PRINT #-1,A$," ",A
30 NEXT
40 PRINT "テープ ヲ マキモトシテ フダサイ"
50 PRINT "OK ナラ RETURN KEY ヲ オシテ フダサイ"
60 INPUT A$
70 RESTORE 200
100 POKE&HFFE6,0:POKE&HFFE7,&HE2
110 FOR I=&HFFE8 TO &HFFFB:READ A$:POKE I,VAL("&
h"+A$):NEXT
120 POKE &HFA08,&HE8:POKE &HFA09,&HFF
150 FOR I=1 TO 5:INPUT #-1,A$:PRINT A$:A:NEXT
200 DATA F5,CD,78,0E,32,1D,FA,E5,2A,E6,FF,77,23,
22,E6,FF,E1,C3,A6,0F
```

テープを録音状態にしてプログラムを実行します。リレーがカッチ、カッチという音をたててデータがSAVEされていきます。そしてしばらくすると、"テープヲマキモトシテクダサイ"のメッセージが表示されますのでテープを巻き戻し、再生状態にして **RETURN** キーを押します。そうしますとテープが回りだしデータを読み込みます。

```
RUN
テープ ヲ マキモトシテ フダサイ
OK ナラ RETURN KEY ヲ オシテ フダサイ
?
F5                245
CD                205
78                120
0E                14
32                50
OK
```

ここで[1]キーを押してページ2に切り換えると今のテーブルフォーマットが表示されます。

```

LLLLLLL F5      ,
      245番LLLLLLL CD
,      ,      205番LLLLLLL 78
      ,      120番しし
LLLLL 0E      ,
      14番LLLLLLL 32      ,
      50番

```

ページ2の表示

","のためにスペースが14個出力されている

E200	9C	9C	9C	9C	9C	9C	46	35	20	20	20	20	20	20	20	20	20
E210	20	20	20	20	20	20	2C	20	20	20	20	20	20	20	20	20	20
E220	20	20	20	20	20	20	32	34	35	0D	9C	9C	9C	9C	9C	9C	9C
E230	43	44	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
E240	2C	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
E250	32	30	35	0D	9C	9C	9C	9C	9C	9C	37	38	20	20	20	20	20
E260	20	20	20	20	20	20	20	20	20	20	2C	20	20	20	20	20	20
E270	20	20	20	20	20	20	20	20	20	20	31	32	30	0D	9C	9C	9C
E280	9C	9C	9C	9C	30	45	20	20	20	20	20	20	20	20	20	20	20
E290	20	20	20	20	2C	20	20	20	20	20	20	20	20	20	20	20	20
E2A0	20	20	20	20	31	34	0D	9C	9C	9C	9C	9C	9C	33	32	20	20
E2B0	20	20	20	20	20	20	20	20	20	20	20	20	20	2C	20	20	20
E2C0	20	20	20	20	20	20	20	20	20	20	20	20	20	35	30	0D	0D

次に20行を

```
20 PRINT#-1,A$;" , ";A
```

に変更して同じように実行してみます。

```

RUN
テーブルヲマキモトメシテフダサイ
OKナラRETURNKEYヲオシテフダサイ
?
F5 245
CD 205
78 120
0E 14
32 50
Ok

```

iiiiiiiF5, 245iiiiiiiCD, 285iiiiiii
ii78, 120iiiiiii0E, 14iiiiiii32,
50

ページ2の表示

```
E200 9C 9C 9C 9C 9C 9C 46 35 2C 20 32 34 35 0D 9C 9C
E210 9C 9C 9C 9C 43 44 2C 20 32 30 35 0D 9C 9C 9C 9C
E220 9C 9C 37 38 2C 20 31 32 30 0D 9C 9C 9C 9C 9C 9C
E230 30 45 2C 20 31 34 0D 9C 9C 9C 9C 9C 33 32 2C
E240 20 35 30 0D 20 20 20 20
```

実行結果から分かるようにカンマ(,)を使うよりもセミコロン(;)の方がデータの長さが短くなります。カンマを使用した場合は2バイトの文字列データだったのがスペースが付加されて16バイトになっています。

ところで、前述の PRINT#-1 文中の";"は省略可能です。先ほどのプログラムの20行を、

```
20 PRINT#-1, A$,"A
```

と変更して実行してみてください。";"を入れたときと同じ結果になるでしょう。

PRINT#-1 を使うときに問題になるのは長いデータは SAVE できないということです。

```
5 CLEAR 300
10 A$="0123456789": DA$=A$+A$+A$+A$+A$+A$+A$+A$+
A$+A$: A$=DA$+DA$
20 PRINT #-1, A$
40 PRINT "テープ ヲ マキモトシテ フダサイ!"
50 PRINT "OK ナラ return key ヲ オシテフダサイ"
60 INPUT A$
70 RESTORE 200
100 POKE &HFFE6,0: POKE &HFFE7,&HE2
110 FOR I=&HFFE8 TO &HFFFB: READ A$: POKE I, VAL(" &
h"+A$): NEXT
120 POKE &HFA08,&HE8: POKE &HFA09,&HFF
150 INPUT #-1, A$: PRINT A$
200 DATA F5, CD, 78, 0E, 32, 1D, FA, E5, 2A, E6, FF, 77, 23,
22, E6, FF, E1, C3, A6, 0F
```

実験的に200バイトの文字列をSAVEして、その後にこれをLOADしてみます。

```
RUN
テープ ヲ マキモトシテ フダサイ!
OK ナラ return key ヲ オシテフダサイ
?
01234567890123456789012345678901
23456789012345678901234567890123
4567890
Ok
```

LLLLLLL01234567890123456789012345
67890123456789012345678901234567
89012345678901

ページ 2 の画面

これで分かるように1つのデータブロックは最大71文字しか使えません。これはバッファが72文字分しかないために起こります。もし長い文字列を SAVE する場合は文字列を分解して SAVE するなどの工夫がいます。

第7章 プリンタ出力

- 7-1 PC-6021
- 7-2 キャラクタ
- 7-3 画面コピーの方法
 - 7-3-1 PC-6021による画面コピー
 - 7-3-2 他のプリンタによる画面コピー
 - 7-3-3 キャラクタのみ画面コピーする方法
- 7-4 PC-8023によるひらがな出力
- 7-5 PC-8023によるグラフィック出力

第7章 プリンタ出力

7-1 PC-6021

PC-6021 は感熱式のプリンタで、PR-1001(PC-8022)と比べて、ビデオコピーの機能を取り除いている他は、機能的に変わるところはありません(もしPC-8022をお持ちの方はPC-6001で使用できます)。

このプリンタは6つのコマンドを備えています。

機 能	略 号	コード(16進)	使 い 方
紙送り(0~255行)	F/F	0C+データ	CHRS(12)+CHRS(n) n=紙送りの行数
右づめ印字	RA	12	CHRS(18)
白黒反転 (グラフィックモード時)	REVERSE	14	※文字の白黒反転は BASICではできない
グラフィックプリント	GS	1D+データ	CHRS(29)+CHRS(L) +CHRS(データ)・・・ CHRS(データ) L=グラフィック印字をする ライン数
モードのリセット	RS	1E	CHRS(30)
拡大印字	US	1F	CHRS(31)

注: 14HのみCHRS命令で送ることができません。白黒反転を行う場合は機械語を使用します。

例 LD A, 14H

CALL 1A16H; プリンタ1文字出力

RET

図7-1 プリンタ・コマンド一覧表

7-2 キャラクタ

PC-6021が印字できるキャラクタは、英記号、英文字、カナに限られており、PC-6001 内部で持っているグラフィック、漢字等はプリンタに出力することができません。

PC-6021 キャラクタ表

	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{		}	~	
	ラ	アイ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
ウ	チ	ツ	テ	ト	ナ	ニ	ノ	ハ	ヒ	フ	ヘ	ホ	マ	ミ	ム
ミ	ム	メ	モ	ヤ	ユ	ヨ	リ	ル	レ	ロ	ワ	ヅ	ヅ	ヅ	ヅ

ただし、ひらがなだけはカタカナに変換して出力されます。

例 LPRINT "あいいうえお"
アイウエオ

PC-6001 には、(N₆₀-BASIC が特殊記号を出力しないように制限しているために、)セントロニクス仕様のプリンタであればどの機種でも接続することができます。しかし、画面 COPY のコントロール(LCOPY)が PC-6021 の機能仕様になっているため、PC-6001 専用となっているプリンタ以外は、基本的には LCOPY 命令は使えません。

N₆₀-BASIC がプリントアウトすることのできるキャラクタを下記に示します。

キャラクタ表示プログラム

```
10 FOR I=32 TO 255 STEP 4:FOR J=0 TO 3
20 LPRINT TAB(8*J);I+J;"=";CHR$(34);CHR$(I+J);C
   HR$(34);
30 NEXT J:LPRINT:NEXT I
```

32=" "	33="!"	34=""	35="#"
36="\$"	37="%"	38="&"	39="?"
40="("	41=")"	42="*"	43="+"
44=","	45="-"	46="."	47="/"
48="0"	49="1"	50="2"	51="3"
52="4"	53="5"	54="6"	55="7"
56="8"	57="9"	58=":"	59=";"
60="<"	61="="	62=">"	63="?"
64="@"	65="A"	66="B"	67="C"
68="D"	69="E"	70="F"	71="G"

72="H"	73="I"	74="J"	75="K"
76="L"	77="M"	78="N"	79="O"
80="P"	81="Q"	82="R"	83="S"
84="T"	85="U"	86="V"	87="W"
88="X"	89="Y"	90="Z"	91="["
92="}"	93="["	94="\"	95="_"
96="`"	97="a"	98="b"	99="c"
100="d"	101="e"	102="f"	103="g"
104="h"	105="i"	106="j"	107="k"
108="l"	109="m"	110="n"	111="o"
112="p"	113="q"	114="r"	115="s"
116="t"	117="u"	118="v"	119="w"
120="x"	121="y"	122="z"	123="{"
124=" "	125="}"	126="~"	127=" "
128=""	129=""	130=""	131=""
132=""	133=""	134="ア"	135="イ"
136="イ"	137="ウ"	138="エ"	139="オ"
140="カ"	141="キ"	142="ク"	143="ケ"
144="コ"	145="サ"	146="シ"	147="セ"
148="ソ"	149="タ"	150="チ"	151="ツ"
152="テ"	153="ト"	154="ナ"	155="ニ"
156="ノ"	157="ハ"	158="ヒ"	159="フ"
160="ボ"	161="ブ"	162="マ"	163="ミ"
164="ム"	165="ヤ"	166="ユ"	167="ヨ"
168="イ"	169="ウ"	170="エ"	171="オ"
172="カ"	173="キ"	174="ク"	175="ケ"
176="コ"	177="サ"	178="シ"	179="セ"
180="ソ"	181="タ"	182="チ"	183="ツ"
184="テ"	185="ト"	186="ナ"	187="ニ"
188="ノ"	189="ハ"	190="ヒ"	191="フ"
192="ボ"	193="ブ"	194="マ"	195="ミ"
196="ム"	197="ヤ"	198="ユ"	199="ヨ"
200="ア"	201="イ"	202="ウ"	203="エ"
204="オ"	205="カ"	206="キ"	207="ク"
208="ケ"	209="コ"	210="サ"	211="シ"
212="セ"	213="ソ"	214="タ"	215="チ"
216="ツ"	217="テ"	218="ト"	219="ナ"
220="ニ"	221="ノ"	222="ハ"	223="ヒ"
224="フ"	225="ボ"	226="ブ"	227="マ"
228="ミ"	229="ム"	230="ヤ"	231="ユ"
232="ヨ"	233="ア"	234="イ"	235="ウ"
236="エ"	237="オ"	238="カ"	239="キ"
240="ク"	241="ケ"	242="コ"	243="サ"
244="シ"	245="セ"	246="ソ"	247="タ"
248="チ"	249="ツ"	250="テ"	251="ト"
252="ナ"	253="ニ"	254="ノ"	255="ハ"

7-3 画面コピーの方法

PC-8001 の場合は、画面コピーをとるには市販の画面コピー用の ROM を購入しなければなりませんが、PC-6001 では LCOPY 命令で簡単に行なえます。

7-3-1 PC-6021による画面コピー

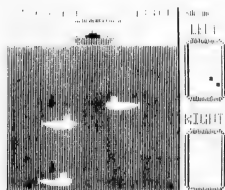
画面コピーの例



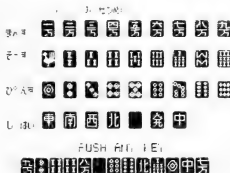
モード 1



モード 2



モード 3



モード 4

画面コピーは256×192のドットで印字されており、このドット数はちょうど、画面のドット数と同じになっています。そのため画面コピーされたものは、カラーを濃淡で表現することができません。

一見、モード3は濃淡で表われているように見えますが、プログラムの的には、カラー対応になっていません。VDGのグラフィックモードを考えれば、この理由が簡単に分かるはずですが、

LCOPYのプログラムで、画面モード3、4では単にビットイメージのデータとしてVRAMの内容をプリンタに転送しているのです。128×192モードでは2ドットで1つの点の色を表わしていましたが、その色によってドットのデータが変わるために濃淡として見えるだけなのです。

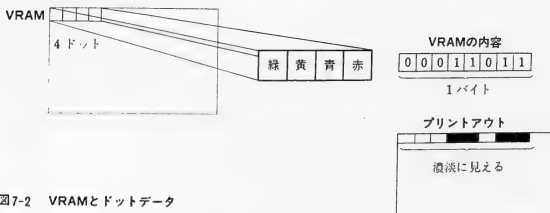


図7-2 VRAMとドットデータ

7-3-2 他のプリンタによる画面コピー

ビットイメージの機能を持つプリンタならば、画面コピーのプログラムをつくることによりコピーが可能になります。

N₆₀-BASIC では、他のプリンタで画面コピーができるように考慮されています。

LCOPY 命令があると、まずFFD2H 番地をコールするようになっています。そこでこの番地を画面コピーのプログラムのアドレスに書き換えて、スタックポインタを細工すれば自分のプリンタにあった LCOPY が可能となります。

例として PC-8023 で画面コピーを行なうプログラムを示します。

画面コピーの例

くくく コンパット ゲーム >>>
 [コンピューターとあなたの手の
 が せう ゲーム です]

おまけ
 ● ● ●
 ● ● ●

おまけ
 ● ● ●
 ● ● ●

1. おまけ...していただきよう(O)の
 きて または よことなりにあるま
 2. おまけ...していただきよう(O)の
 なるまにあるま
3. (おまけ!)のとき...●
 ●がめんにひょうし(O)をうやしらけ
 しだいしとき...X=B, Y=B
 一人だのへんせい—せんかん...1,
 くま...1,しつせんかん...1,
 しし...2,くまかん...3

*** とく 手生 おしてくさい ***

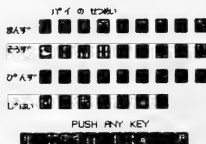
モード 1

麻雀ゲーム
 for
 PC
 6001
 BY RAY KAZUTO & YELLOW PANTHER

モード 2



モード 3



モード 4

PC-8023 による画面コピーのプログラム

```

10 CLEAR 300,&HDF00-1
100 FOR I=&HDF00 TO &HDFFF:READ A$:POKE I,VAL("&H
"+A$):NEXT I
110 AD=&HFFD2:POKE AD,&HC3:POKE AD+1,&H25:POKE A
D+2,&HDF
1000 DATA 1b,53,30,31,39,32,1b,54,30,30,0d,1b,54,
31,36,0d
1010 DATA 0a,1b,41,0a,3e,06,11,00,df,f5,1a,4f,cd,
16,1a,13
1020 DATA f1,3d,20,f5,c9,e5,3a,91,fd,c6,02,67,2e,
1f,cd,0a
1030 DATA 13,da,70,df,06,20,e5,cd,14,df,3e,c0,f5,
7e,cd,16
1040 DATA 1a,11,20,00,19,f1,3d,20,f3,3e,0b,11,06,
df,cd,19
1050 DATA df,e1,2b,10,e1,3e,03,11,11,df,cd,19,df,
3e,0a,06
1060 DATA 06,cd,1c,1a,10,fb,e1,d1,c9,00,00,00,00,
00,00,00
1070 DATA 06,20,e5,cd,14,df,3e,10,f5,e5,25,25,7e,
e6,40,eb
1080 DATA e1,e5,c5,0e,00,cc,a4,df,c4,c9,df,c1,e1,
11,20,00
1090 DATA 19,f1,3d,20,e3,3e,0b,11,06,df,cd,19,df,
e1,2b,10
1100 DATA d1,c3,55,df,e5,7e,d5,cd,a0,14,eb,06,00,
09,3e,04
1110 DATA d3,93,d1,1a,0f,7e,30,01,2f,cd,16,1a,e1,
0c,79,fe
1120 DATA 0c,20,e1,3e,05,d3,93,af,c9,e5,c5,79,0f,
0f,e6,03
1130 DATA 2f,c6,04,4f,7e,0d,28,04,0f,0f,18,f9,e6,
03,4f,06
1140 DATA 00,21,4f,23,09,7e,cd,16,1a,c1,e1,0c,79,
fe,0c,20
1150 DATA d8,c9,00,00,00,00,00,00,00,00,00,00,00,
00,00,00

```

7-3-3 キャラクタのみ画面コピーする方法

キャラクタコピーだけでよければ、BASIC で簡単につくることができます。

キャラクタ・コピー プログラム

```

10000 P=PEEK(&HFD90)+1:S=0:E=15
10010 IF P=1 THEN AD=PEEK(&HFA60):AD=(AD-2)*256:GO
TO 10030
10020 AD=&HE200:AD=AD+((2-P)*&H2000)
10030 FOR I=S TO E:FOR J=0 TO 31
10040 DA=PEEK(AD+3*I+J):LPRINT CHR$(DA):;NEXT J:L
PRINT :NEXT I
10050 RETURN

```

実行例

LIST

```
10000 P=PEEK(&HFD90)+1:S=0:E=15
10010 IF P=1 THEN AD=PEEK(&HFA60
):AD=(AD-2)*256:GOTO 10030
10020 AD=&HE200:AD=AD+((2-P)*&H2
000)
10030 FOR I=S TO E:FOR J=0 TO 31

10040 DA=PEEK(AD+32*I+J):LPRINT
CHR$(DA);:NEXT J:LPRINT :NEXT I
10050 RETURN
Ok
gosub 10000

1 COLOR CLOAD GOTO LIST RUN
```

このプログラムはサブルーチン形式になっています。Pにコピーしたいページ、Sにプリント開始行、Eに最終行を入れて GOSUB 10010 でコールすれば、そのページの S と E で指定した範囲のコピーを行ないます。

7-4 PC-8023によるひらがな出力

PC-8023 には、“ひらがな”を印示する機能がありますので、これを使ってひらがなを出力させてみましょう。

ひらがな出力プログラム

```
10 REM PC-8023 ひらがな
20 CLEAR 300,&HDF00-1
100 FOR I=&HDF00 TO &HDF70:READ A$:POKEI,VAL("&h
"+A$):NEXT I
110 POKE&HFFD1,&HDF:POKE&HFFD0,&H0:POKE&HFFCF,&H
C3
120 POKE&HFFE4,0
1000 DATA f5,3a,50,fa,fe,01,20,02,f1,c9,f1,33,33,
f5,fe,09
1010 DATA 20,0e,3e,20,cd,c7,26,3a,57,fa,e6,07,20,
f4,f1,c9
1020 DATA f1,f5,d6,0d,20,06,30,07,3a,57,fa,3c,32,
57,fa,f1
```

```

1030 DATA f5,c5,fe,86,38,22,fe,a0,38,04,fe,e0,38,
    1a,ee,20
1040 DATA f5,3a,e4,ff,b7,20,0d,3e,1b,cd,16,1a,3e,
    26,cd,16
1050 DATA 1a,32,e4,ff,f1,c3,29,1a,f5,3a,e4,ff,b7,
    28,0e,3e
1060 DATA 1b,cd,16,1a,3e,24,cd,16,1a,af,32,e4,ff,
    f1,c3,29
1070 DATA 1a

```

```

10 REM あいうえお
20 REM かきくけこ
30 REM さしすせそ
40 REM たちつてと
50 REM なにぬねの
60 REM はひふへほ

```

このプログラムは DF00～DF70H までを使っています。ページ数の指定によってはリロケートの必要があります。

7-5 PC-8023によるグラフィック出力

つぎに、PC-8023 にグラフィックキャラクタを、出力させてみましょう。

PC-6001 のグラフィックパターンは PC-8023 にはないものが多いため、プリンタに出力しようとする、どうしてもビットイメージで出力するしか方法がありません。また、キャラクタによっては、コントロールコードと重なる部分があるので、これらの処理が必要となります。プログラム自体は簡単なのですが、ビットイメージのデータでかなりメモリを専有します。

グラフィックパターンは 8×12ドットにしなければならないのですが、プログラムの簡素化のために 8×8ドットで行なっていますので、すこし見づらくなります。

なおこのプログラムには、ひらがな出力プログラムも入れておきます。

```

10 CLEAR 300,&HDDFF
20 FOR I=&HDE00 TO &HDFCF:READ A$:POKE I,VAL("&h"+A$):NEXT
30 POKE &HFFD1,&HDE:POKE &HFFD0,&H0:POKE &HFFCF,&HC3
1000 DATA f5,3a,58,fa,fe,01,28,02,f1,c9,f1,33,33,
    f5,fe,09
1010 DATA ca,e0,26,d6,0d,ca,fa,26,da,fa,26,3a,57,
    fa,3c,32
1020 DATA 57,fa,f1,e5,c5,cd,8b,10,da,4c,1a,05,28,
    33,fe,86
1030 DATA 38,23,fe,a0,38,04,fe,e0,38,1b,c3,c0,df,
    3e,1b,cd
1040 DATA 16,1a,3e,26,cd,16,1a,f1,cd,16,1a,3e,1b,
    cd,16,1a

```

1050 DATA 3e,24,c3,29,1a,fe,80,da,29,1a,fe,86,d2,
 29,1a,d6
 1060 DATA 60,e5,d5,26,00,6f,29,29,29,11,90,de,19,
 11,87,de
 1070 DATA 06,06,1a,cd,16,1a,13,10,f9,06,08,7e,cd,
 16,1a,23
 1080 DATA 10,f9,d1,e1,c1,f1,c9,1b,53,30,30,30,38,
 00,00,00
 1085 DATA 00,00,00,00,00,00,00,00,00,00,00,7f,29,29,
 29,ff,00
 1090 DATA 80,46,20,1f,20,46,80,00,44,24,1c,ff,0c,
 32,c1,00
 1100 DATA c4,34,0c,ff,0c,34,c4,00,88,cc,aa,f9,aa,
 cc,88,00
 1110 DATA 40,40,44,7f,44,40,40,00,00,7f,49,49,49,
 7f,00,00
 1120 DATA 24,33,2c,fe,2c,22,20,00,ff,09,09,0f,09,
 09,ff,00
 1130 DATA 3e,2a,3e,68,aa,ff,2a,20,00,cc,3a,09,8a,
 7c,08,00
 1140 DATA 2c,fe,29,1c,80,7f,28,10,f9,a9,ad,ab,a9,
 f9,00,00
 1150 DATA 10,14,14,fe,11,10,10,00,20,19,0d,0b,09,
 f9,00,00
 1160 DATA 00,02,3e,02,7e,42,62,00,18,18,18,1f,1f,
 18,18,18
 1170 DATA 18,18,18,f8,f8,18,18,18,18,18,ff,ff,
 18,18,18
 1180 DATA 00,00,00,ff,ff,18,18,18,18,18,ff,ff,
 18,18,18
 1190 DATA 00,00,00,ff,ff,00,00,00,18,18,18,18,18,
 18,18,18
 1200 DATA 00,00,00,f8,f8,18,18,18,18,18,18,f8,f8,
 00,00,00
 1210 DATA 00,00,00,1f,1f,18,18,18,18,18,18,1f,1f,
 00,00,00
 1220 DATA 00,42,24,18,18,24,42,00,84,44,24,1f,24,
 44,84,84
 1230 DATA 0e,0a,0a,ff,0a,0a,0e,00,20,18,00,ff,00,
 00,10,20
 1240 DATA 30,9c,de,ff,de,9c,30,00,0e,3f,7e,fc,7e,
 3f,0e,00
 1250 DATA 1c,9c,df,ff,df,9c,1c,00,00,1c,3e,7f,3e,
 1c,00,00
 1260 DATA 3c,42,81,81,81,81,42,3c,3c,7e,ff,ff,ff,
 ff,7e,3c
 1270 DATA fe,fe,30,02,ee,20,f5,c3,3d,de,00,00,00,
 00,00,00

第8章 N₆₀-BASICの命令分析

- 8-1 N₆₀-BASICとN-BASICの命令比較
- 8-2 いろいろな命令
 - 8-2-1 LINE
 - 8-2-2 PSET, PRESET
 - 8-2-3 COLOR
 - 8-2-4 CLS
- 8-3 色のつけ方
- 8-4 N₆₀-BASICにない命令をある命令で
代用する

第8章 N₆₀-BASIC の命令分析

8-1 N₆₀-BASIC と N-BASIC の命令比較

N₆₀-BASIC およびN-BASIC は同じマイクロソフト社製の BASIC です。基本的な命令は同じになっていますが、ハードや、使用目的の違いから、画面処理やサウンド機能で大きな差異が見られます。

(1) N₆₀-BASIC, N-BASIC と同一機能

ABS	AND	ASC	CHR\$
CLEAR	CLOAD	CLOAD?	CONT
COS	CSAVE	CSRLIN	DATA
DEFFN	DIM	END	EXP
FOR	FRE	GOSUB	GOTO
IF	INKEY\$	INP	INPUT
INPUT#	INT	KEY	LEFT\$
LEN	LET	LIST	LLIST
LOG	LPOS	LPRINT	MID\$
NEXT	NEW	NOT	ON GOSUB
ON GOTO	OR	OUT	PEEK
POKE	PRINT	PRINT#	READ
REM	RESTORE	RETURN	RIGHT\$
RND	RUN	SGN	SIN
SPC	SQR	STEP	STOP
STR\$	TAB	TAN	THEN
VAL			

(2) N₆₀-BASIC と N-BASIC ではパラメータが異なるもの

COLOR	CONSOLE	LINE	LOCATE
POINT	PRESET	PSET	USR
TIME			

(3) N₆₀-BASIC ではけずられたもの

ATN	AUTO	BEEP	CDBL
CINT	CSNG	DEFDBL	DEFINT
DEFSNG	DEFSTR	DEFUSR	DELETE
ELSE	EQV	ERASE	ERL
ERR	ERROR	FIX	GET@
HEX\$	IMP	INPUT\$	INSTR
KEYLIST	LINEINPUT	LPRINTUSNG	MOD
MON	MOTOR	OCT\$	ON ERROR GOTO
PRINT USING	PUT@	RENUM	SPACES\$
SWAP	TERM	TIMES\$	TROFF
TRON	VARPTR	WAIT	WIDTH
XOR			

(4) N₆₀-BASIC にしかないもの

CLS	EXEC	LCOPY	PAINT
PLAY	SCREEN	STICK	STRIG
SOUND			

(1)に属する部分は BASIC の基本的な命令で使い方に他機種との差異はありません。

(2)の部分はおもにハードウェアの違いによるもので、PC-8001 のグラフィックは160×100 ドット、カラー8色が使えましたが、CRT コントローラの LSI の違いから、PC-6001 ではグラフィックモードが3種類選べ、おのおのモードで使用できる色の数が異なりますので、プログラムを移植するときに注意が必要です。

N-BASIC では整数、単精度、倍精度の3種類あり、変数名を型指定できましたが、N₆₀-BASIC では数値型が1種類しかないために、数値型宣言の命令である DEFSNG、DEFINT 等の命令や数値型の相互変換の命令である CINT 等の命令もありません。

エラー処理のための命令もすべてカットされています。この部分は、プログラムでエラーさえ出さなければ必要ありませんので、別になくても困りません。

TRON、RENUM、AUTO 等のコマンドもなくなっていますが、これらはプログラムのエディット時およびデバッグ中にしか使いませんので、特になくても困りません。

けずられた命令で1番困るのは、ゲーム等で多用する GET@、PUT@で、これらがいないために、絵を高速で動かす場合はどうしても機械語を使用することになります。

最後に、N-BASIC のプログラムを N₆₀-BASIC に変換するプログラムについて考えてみましょう。

PC-8001 の上位コンパチブルに設計してある PC-8801 では、PC-8001 から PC-8801 へのテキストコンバータ(プログラムの自動変換プログラム)は簡単につくることはできますが、

ハード的にまったく異なる PC-6001 と PC-8001 では、実用になるテキストコンバータをつくることは不可能に近いものになります。たとえば BASIC の基本命令だけで書かれたプログラム(ホームコンピュータプログラム集・システムソフト, BASIC COMPUTER GAMES・アスキー等)ならばコンバート可能です。しかし、もしもとなるプログラムが80文字×25行に合わせて画面表示するようにしてあるとすれば、32文字×16行しか表示できない PC-6001 では画面の表示がおかしくなります。結局このようなプログラムは、人手による手直しが必要です。

8-2 いろいろな命令

基本的な命令の説明は入門書等に譲るとして、ここでは N-BASIC とくに使い方が異なる命令について説明します。

N-BASIC と比較して大幅に異なるのが、画面まわりの命令で、画面出力の命令としては、文字出力関係の PRINT, LOCATE, TAB, SPC等があり、また、グラフィック関係としては、LINE, PSET, PRESET, PAINT 等があります。文字出力の命令は32文字×16行に変更になった以外は、変わる所がありませんのでとくに説明はしません。

ここでは各章で説明をしなかった命令について説明します。

8-2-1 LINE

N-BASIC では画面が40文字と80文字とでは X 軸の座標のパラメータは 0~79 から 0~159 へ変化しましたが、N₈₀-BASIC においては、グラフィックのドットに関係なく 0~255(X 軸), 0~191(Y 軸)の固定となっています。そのために、モード 1, 2, 3 で使用する場合は補正する必要があります。たとえばセミグラフィックモードでは 64×48 ドットですから、もし、座標 (20, 10) から (40, 20) に直線を書かせる場合は、

LINE(20*256/64, 10*192/48)-(40*256/64, 20*192/48)

X 軸の補正 Y 軸の補正

になります。また、N-BASIC においては PSET, PRESET のどちらかを指定できましたが、N₈₀-BASIC では指定できませんので、点を消したい場合はバックと同じ色を指定することによって行ないます。

また N-BASIC ではキャラクタを使って線をかいたり、箱をかくことが可能でしたが N₈₀-BASIC では、その機能はオミットされています。

LINE(0, 0)-(10, 0), "♥"

N-BASIC の場合

FOR I=0 TO 10: LOCATE I, 0: PRINT "♥";: NEXT I N₈₀-BASIC の場合

上記のように FOR~NEXT で代用しなければなりません。また、LINE を使用してのリンク機能もなくなっています。

8-2-2 PSET, PRESET

座標が 0 ~ 255, 0 ~ 191 で扱われますので、LINE 命令と同じように補正しなければいけません。たとえば、モード 2 において PSET(0, 0) ~ PSET(3, 3) までは同じ位置に点を表示します。なお、PSET のカラーの指定が N-BASIC と異なっています。

PSET(X, Y, C).....N-BASIC

PSET(X, Y), C.....N₈₀-BASIC

座標の補正値を下に示しておきます。

モード	X	Y
1	8	12
2	4	4
3	2	1
4	1	1

図 8-1 モードと座標の補正値

使い方

モード 2 (セミグラフィック) において (20, 10) の位置に点を表示するとき

PSET(20*4, 10*4)

8-2-3 COLOR

N-BASIC では 0 ~ 7 までの 8 色を指定できますが、N₈₀-BASIC においてはモードによって指定できる色の種類が異なります。

また、COLOR の第 2 パラメータと第 3 パラメータの意味が変わっています。

N-BASIC の第 2, 第 3 パラメータに当たる部分が、N₈₀-BASIC にはありません。COLOR に関しては CRTC (画面コントローラの LSI) の違いにより、N-BASIC と N₈₀-BASIC では完全に対応することができなくなっていますので、プログラムの移植には、最も注意を必要とします。

N₈₀-BASIC ではモードによって色の指定が変わります。

COLOR F, B, C

F=FORE GROUND COLOR (文字, 点の色)

B=BACK GROUND COLOR (背景の色)

C=COLOR SET (色の組み合わせ)

モード1 (テキストモード)

第1パラメータ	第2パラメータ	第3パラメータ	色
1	×	1	緑
2	×	1	緑の反転
3	×	1	オレンジ
4	×	1	オレンジの反転
1	×	2	オレンジ
2	/	2	オレンジの反転
3	/	2	緑
4	/	2	緑の反転

※色は緑とオレンジの2色で第3パラメータによって入れ換わるだけです
図8-1-1

モード2 (セミグラフィック)

第1パラメータ	第2パラメータ	第3パラメータ	色
0	×	1	黒
1	/	1	緑
2	/	1	黄
3	/	1	青
4	/	1	赤
5	×	1	白
6	×	1	水色
7	×	1	紫
8	×	1	オレンジ
0	×	2	黒
1	×	2	白
2	×	2	水色
3	×	2	紫
4	×	2	オレンジ
5	×	2	緑
6	×	2	黄
7	×	2	青
8	×	2	赤

図8-1-2

モード3 (128×192 カラーグラフィック)

第1パラメータ	第2パラメータ	第3パラメータ	色
1	1	1	緑
2	2	1	黄
3	3	1	青
4	4	1	赤
1	1	2	白
2	2	2	水色
3	3	2	紫
4	4	2	オレンジ

COLORで指定の後、最初のCLS命令で画面全体が第2パラメータの色になる

図8-1-3

モード4 (256×192グラフィック)

第1パラメータ	第2パラメータ	第3パラメータ	色
1	0	1	黒地に緑の グラフィック
0	1	1	緑地に黒の グラフィック
1	0	2	黒地に白の グラフィック
0	1	2	白地に黒の グラフィック

図8-1-4

これら COLOR の F と B のパラメータは規定の範囲を超えて指定した場合は最大値、または、最小値にセットされます。ところがモード2のとき、この処理にBUGがあり最大値がうまくセットされません。

モード2のBUG

SCREEN 2, 1, 1 : COLOR 9 : CLS

8-2-4 CLS

N₆₀-BASIC の CLS は、N-BASIC の PRINT CHR\$(12) と同じ命令(N₆₀-BASIC でも PRINT CHR\$(12) で画面クリア可能)ですが、N₆₀-BASIC では CONSOLE で指定した範囲でしか画面がクリアされません。

```
10 CLS:CONSOLE 5,5,1
20 FOR I=0 TO 14:LOCATE 0,I:PRINT I:NEXT I:CLS
```


0
1
2
3
4
Ok

10
11
12
13
14
1 COLOR CLOAD GOTO LIST RUN

実行結果で分かるように CONSOLE で指定した範囲しかクリアされていません。この方が便利な場合が多いので別に困りませんが、もし画面全体をクリアしたい場合は一時的に CONSOLE を元に戻すことで可能です。

N60-BASIC で一番困ることは、グラフィックモードのときに画面に OR でドットを書かせますので一度書かれた上に重ね書きされ、画面が見苦しくなります。

```
10 SCREEN 3,2,2:CLS
20 LOCATE 0,8:PRINT"てすと"
30 LOCATE 0,8:PRINT"あいう"
```

あああああ

これを解決するには、ドットとバックの色を同じにして塗ればよく、これには LINE の BF 命令を使います。

```
10 SCREEN 3,2,2:CLS
20 LOCATE 0,8:PRINT"てすと"
25 COLOR 1,1,1:LINE(0,96)-(50,107),1,BF:COLOR 2
30 LOCATE 0,8:PRINT"あいう"
```

あああいう

8-3 色のつけ方

画面に色を付けるには V-RAM およびアトリビュートにデータを書くことによって行ないますが、モードによってその方法が異なりますので、その使い方について説明します。

今、E200H 番地の値が 61H とすると、モードによってどのように表示されるでしょうか。

VRAMの内容

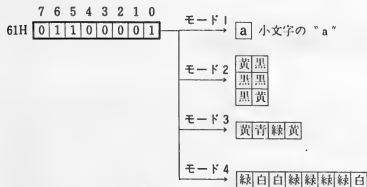


図 8-2 モードと表示の差

※COLORはすべて初期値とする

このように同じデータでもモードによって大きく違うことが分かります。

(1) モード 1

文字とバックの色を変える信号はアトリビュートの bit 1 (CSS) と bit 0 (INV) で行ないます。

CSS 信号が文字の色を、INV がバックの色を決定します。

CSS→0=緑

→1=オレンジ

INV→0=黒

→1=緑またはオレンジ(CSS できまる)

次のプログラムを実行すれば CSS, INV 信号と色の関係がよく分かるはずです。

```

10 SCREEN 1,2,2:CLS
20 POKE &HE200,&H61:LOCATE 5,8:PRINT "CSS   INU
"
30 FOR I=0 TO 3:A=PEEK(&HE000)AND&HFC:POKE &HE0
00,A OR I
40 IF I AND 2 THEN LOCATE 6,9:PRINT "1":GOTO 60
50 LOCATE 6,9:PRINT "0"
60 IF I AND 1THEN LOCATE 12,9:PRINT "1":GOTO 80
70 LOCATE 12,9:PRINT"0"
80 FOR J=0 TO 300:NEXT J:NEXT I
90 GOTO 30

```

a

```

CSS   INU
  1     1

```

2 COLOR CLOAD GOTO LIST RUN

(2) モード2

このセミグラフィックは1キャラクタごとにしか色を定義できません。これはPC-8001の場合とよく似ています。

$\underbrace{D_7 \ D_6}_{\text{色を設定する}} \quad \left. \begin{array}{c} D_5 \ D_4 \\ D_3 \ D_2 \\ D_1 \ D_0 \end{array} \right\} \text{ドットを設定する}$

CSS	D ₇	D ₆	色
0	0	0	緑
	0	1	黄
	1	0	青
	1	1	赤
1	0	0	白
	0	1	水色
	1	0	紫
	1	1	オレンジ

図8-3 セミグラフィックのカラー設定

```

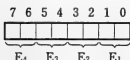
10 SCREEN 2,2,2:CLS
20 LOCATE 4,8:PRINT"DB 7 6 5 4 3 2 1 0 CSS"
30 FOR CS=0 TO 1:A=PEEK(&HE000)OR CS*2:POKE&HE0
  00,A:LOCATE 24,9:PRINTCS
40 FOR I=0 TO 255
50 C=I:FOR J=7 TO 0 STEP -1
60 IF C>=2^J THEN LOCATE 21-J*2,9:PRINT "1":C=C
  -2^J:GOTO 80
70 LOCATE 21-2*J,9:PRINT"0"
80 NEXT J:POKE &HE200,I:FOR J=0 TO 1:NEXTJ:NEXT
  I:NEXT CS:GOTO 30

```

このプログラムでデータと表示を見ることができます。

(3) モード3

CSS とデータによって色を変えますが、CSS が、1 バイトごとでしか定義できませんので
 キャラクタ内では4色しか使えません。



この2ドットの組み合わせで色をきめる

CSS	D ₇ , D ₅ D ₃ , D ₁	D ₆ , D ₄ D ₂ , D ₀	色
0	0	0	緑
	0	1	黄
	1	0	青
	1	1	赤
1	0	0	白
	0	1	水色
	1	0	紫
	1	1	オレンジ

図8-4 モード3のカラー設定

カラーを調べるプログラム

```

10 SCREEN 3,2,2:CLS:COLOR 2
20 LOCATE 0,8:PRINT"7 6 5 4 3 2 1 0":LOCATE0,10
  :PRINT"CSS"
30 FORCS=0 TO 1:A=PEEK(&HE000) OR CS*2:POKE&HE0
  00,A:COLOR 1,1,1
35 LINE(2,132)-(40,143),1,BF:LOCATE 1,11:COLOR
  2:PRINT CS
40 FOR I=0 TO 255
50 C=I:FOR J=7 TO 0 STEP -1
60 IF C>=2^J THENLOCATE 14-J*2,9:PRINT"1":C=C-2

```

```

^J:GOTO 80
70 LOCATE 14-2*J,9:PRINT"0"
80 NEXT J:POKE &HE200,I:FOR J=0 TO 100:NEXT
90 LOCATE 0,9:COLOR1,1,1:LINE(0,100)-(255,119),
1,BF:COLOR 2
100 NEXT I:NEXT CS:GOTO 30

```

(4) モード4

CSS とデータ・ビットに対応した部分の色が変わります。このモードはプログラムのには一番使いやすいといえます。

CSS	データの各ビット	色
0	0	黒
	1	緑
1	0	黒
	1	白

図8-5 モード4とカラー設定

色を調べるプログラム

```

10 SCREEN 4,2,2:CLS:COLOR 1
20 LOCATE 4,8:PRINT"DB 7 6 5 4 3 2 1 0 CSS"
30 FOR CS=1 TO 2:COLOR,,CS
40 FOR I=0 TO 255
45 COLOR 0,0,CS:LINE(0,100)-(240,119),0,BF:COLO
R1:LOCATE24,9:PRINTCS-1
50 C=I:FOR J=7 TO 0 STEP -1
60 IF C>=2^J THEN LOCATE 21-J*2,9:PRINT"1":C=C-
2^J:GOTO 80
70 LOCATE 21-2*J,9:PRINT"0"
80 NEXT J:POKE &HE200,I:FOR J=0 TO 100:NEXT J:N
EXT I:NEXT CS:GOTO 30

```

このような VRAM の操作は機械語レベルで行なうと、BASIC と比較してきめ細かな操作ができます。

8-4 N₆₀-BASIC にない命令をある命令で代用する

N₆₀-BASIC では削られた命令の内、一部は、他の命令を組み合わせで代用することができます。N-BASIC のプログラムを移植するときの参考にして下さい。

N-BASIC		N ₆₀ -BASIC
AUTO	→	} 第9章 EXEC と USR 参照
KEY LIST	→	
BEEP	→	PRINT CHR\$(7) または EXEC &H1BCD
BEEP0	→	EXEC &H1B60
BEEP1	→	SOUND 0, 85 : SOUND 1, 0 : SOUND 7, 62 : SOUND 8, 7
MOTOR 0	→	EXEC &H1B49
MOTOR 1	→	EXEC &H1B4B
X MOD Y	→	$X - \text{INT}(X/Y) * Y$
MON	→	付録参照

第9章 EXECとUSR

9-1 モニタ

9-2 EXECとUSR

9-2-1 EXEC

9-2-2 EXECの応用

9-2-3 USR

9-3 引数

9-3-1 数値型

9-3-2 文字型

9-4 BASICを機械語で

9-4-1 ファンクションキー・イニシャライズ

9-4-2 KEY LIST

9-4-3 日本語エラーメッセージ

第9章 EXEC とUSR

9-1 モニタ

PC-6001 にはモニタがありませんので、機械語を使用したいときは PEEK, POKE の命令を用いて、メモリ内容を操作しなくてはなりません。これでは、機械語プログラムをつくる時に非常に不便です。

巻末に、タイニー・モニタプログラムを載せましたが、本格的に機械語を使ったり、BASIC の内部を知りたいときには、やはり、モニタ・ROM が必要でしょう。

暴走するたびに、LOAD していたのではプログラミングどころではありませんし、他の BASIC プログラムを実行することができないからです。

各社から、PC-6001 用のモニタ、アセンブラが発売されていますので、是非、利用されたいでしょう。(PAPA-MONITOR・システムソフト、SEAM60-アセンブラ・アスキー コンシューマー プロダクツ等)

機械語を打ちこむ簡単なプログラムは下記のとおりです。

```
100 INPUT "ADR";D$
110 D=VAL("&h"+D$):IF D<0 THEN D=65536+D
120 FOR I=D TO 65535:INPUT A$:POKE I,VAL("&h"+A$)
    :NEXT I
```

9-2 EXEC と USR

N₈₀-BASIC においては機械語のプログラムをコールしたい場合は、EXEC または USR を使います。

EXEC と USR の違いは、EXEC が機械語プログラムをコールするだけなのに対して、USR は機械語プログラムにデータを受渡すことができ、また、機械語プログラムから BASIC へデータの受渡しもできます。

さて、EXEC と USR の使い方を実例を交えて説明することにしましょう。

9-2-1 EXEC

EXEC は直接、機械語プログラムに対して直接データの受渡しはできません。

使い方は、EXEC の後に実行する機械語のアドレスを与えます。このアドレスは10進、16進、変数、式のいずれも使用可能です。

機械語のプログラムに RET 命令があると BASIC へ戻ります。このときレジスタの値はどのような値でもかまいません。すなわち、EXEC や USR では BASIC から飛んてくる前にレジスタの退避が済んでいますので、機械語プログラムでレジスタを保存する必要はありません。

10 EXEC &H1DFB

上のプログラムを RUN して下さい。画面がクリアされたはずですが。

これは1DFB 番地が画面クリア (CLS 命令) の処理ルーチンで、ここをコールすれば当然のことながら画面がクリアされるわけです。

このように EXEC は機械語プログラムをコールするだけで、機械語プログラムとのデータの受渡しはできないようになっていますが、使い方によっては、データを受渡すことが可能です。

それはデータを POKE 文でメモリに書き込んでおき、機械語プログラムに飛ばした後、そのメモリの内容を機械語ルーチンで処理します。その後、結果をメモリにストアし、BASIC に戻り、PEEK 文で取り出すことによって BASIC のデータとして使用する方法です。

例として STICK の命令を EXEC で処理したプログラムを挙げます。なお、ページ指定は32KB、16KB とも 2 を指定しておいて下さい。

```

10 CLEAR 100,&HDDFF
20 FOR I=&HDE00 TO &HDE0D
30 READ DA$:POKE I,VAL("&h"+DA$)
40 NEXT I
50 A=0:POKE &HDE10,A
60 EXEC &HDE00
70 B=PEEK(&HDE10)
80 PRINT B
90 GOTO 50
100 DATA 3a, 10, de, cd, 39, 22, cd, 41, 07, 7b, 32, 10, de,
    c9

```

50行の変数Aはジョイスティックの番号を表わします。

A = 0 : キーボード

A = 1 : ジョイスティック 1

A = 2 : ジョイスティック 2

ジョイスティックをお持ちでない方は、必ずA = 0 にしておいて下さい。

```

3A10DE  STICK: LD  A, (0DE10H) ジョイスティックの No. を Acc に入れる
CD3922  CALL 2239H ジョイスティックの方向を調べる
CD4107  CALL 0741H FAC-1の値を整数に直す
7B      LD  A,E
3210DE  LD  (0DE10H),A Acc の値を DE10H番地に格納する
C9      RET

```

PEEK, POKE でのデータ受渡しではデータの数が増えるほど、BASIC での処理に時間がかかり過ぎます。これでは機械語を使う意義がありませんし、テクニック的にも幼稚過ぎます。しかし、工夫しだいで簡単にデータを受渡せることが分かります。

9-2-2 EXEC の応用

N₆₀-BASIC 内部で EXEC がどのように処理されているか説明します。

EXEC 命令があると、まず機械語の飛び先のアドレスを調べて DE レジスタに入れます。次に BASIC テキストのポインタである HL レジスタをスタックに入れます。そして機械語プログラムからの戻り先になる 39B8H 番地をスタックに入れ、DE と HL の値を交換し、HL で指される番地にジャンプします。

例

```
10 EXEC&HDE00:END
```

としたとき、

				10	EXEC	&	H	
8400	00	0F	84	0A	00	A5	26	48
	D	E	0	0	:	END		
8408	44	45	30	30	3A	80	00	00

スタック

840CH	②
39B8H	③

- ① "&HDE00"を整数に直してDEに入れる
- ② HLの値(840CH番地をさしている)をスタックに入れる
- ③ スタックに39B8Hを入れる(機械語からの戻り先)
- ④ DEとHL交換(EX DE, HL)
- ⑤ HL番地へジャンプする(JP (HL))

図9-1

となります。

したがって、機械語ルーチン内で、スタック中の HL の値を操作して、EXEC 文で直接パラメータを渡すことができます。

実際のパラメータの与え方としては、EXEC [アドレス], [パラメータ] の型になります。このような型で EXEC を実行すると、スタックに入った HL の値はカンマのあるアドレスになっていますから、機械語プログラムで、この HL の値を取り出して、後のパラメータを取り込むことによりデータをもらうことができます。

例として行番号を自動的に発生する AUTO のコマンドを作ってみました。

```

10 CLEAR 300,&HDF3F
20 FOR I=&HDF40 TO &HDFFF:READ A$:POKEI,VAL("&H
   "+A$):NEXT I
100 DATA E1,21,0A,00,22,F4,DF,22,F6,DF,E1,2B,D7,
   28,26,FE
110 DATA 2C,C2,EA,03,D7,FE,2C,28,09,CD,5F,07,ED,
   53,F4,DF
120 DATA 28,13,CF,2C,28,0F,CD,5F,07,ED,53,F6,DF,
   C2,EA,03
130 DATA 7B,B2,CA,55,07,3E,C3,32,93,FF,32,9C,FF,
   21,C5,DF
140 DATA 22,94,FF,21,AC,DF,22,9D,FF,CD,58,10,CD,
   01,27,CD
150 DATA 2D,27,2A,F4,DF,E5,CD,A1,3A,3E,20,CD,C7,
   26,CD,F9
160 DATA 28,38,2F,D7,CD,1E,05,47,D1,C3,78,04,F1,
   CD,DE,DF
170 DATA ED,5B,F4,DF,2A,F6,DF,19,38,19,11,FA,FF,
   E7,30,13
180 DATA 22,F4,DF,18,CD,F1,3E,0D,CD,C7,26,3E,0A,
   CD,C7,26
190 DATA 18,DE,E1,3E,C9,32,93,FF,32,9C,FF,C3,57,
   04,23,EB
200 DATA 62,6B,7E,23,B6,C8,23,23,23,AF,BE,23,20,
   FC,EB,73

```

210 DATA 23, 72, 18, EC, 00, 00, 00, 00, 00, 00, 00, 00,
00, 00, 00

使い方

EXEC&HDF40, 行番号, 増分

行番号に始まり, 増分ずつ加算された行番号を発生します。このコマンドは STOP を押すことにより解除されます。

例 EXEC&HDF40, 100, 10 100 番から10 番おきに行番号を発生。

9-2-3 USR

USR は引数を持って機械語プログラムをコールします。この USR は N-BASIC と比較して 2 つほど機能が低下しています。

ひとつは、機械語プログラムのアドレスをセットするとき、N-BASIC では DEFUSR でセットできたのに対して、N₆₀-BASIC では POKE 文を使い、FAEBH 番地、FAECH 番地にセットしなければならないことです。また、N-BASIC では 10 種類定義できたのに対して、ひとつしか定義できません。

しかし、このあたりの機能を補なうために EXEC の命令があります。大体において機械語を単にサブルーチンとして使う例が多く、引数を受渡すような使い方は極めて少ないと言えます。それならば定義する必要のない EXEC の方が USR より使いやすいことになります。

さて、もうひとつは受渡す引数の型が数値型しかできないことです。N-BASIC では、数値型(整数, 単精度実数, 倍精度実数)と文字型が使用可能でした。N₆₀-BASIC では文字型は TM Error になり、使うことができません。

```
10 CLEAR 300,&HDDFF
20 POKE&HDE00,&HC9
30 POKE&HFAEB,0
40 POKE&HFAEC,&HDE
50 INPUT X
60 A=USR(X)
70 PRINT A:GOTO 50
```

このプログラムは、機械語プログラムではなにもしないで BASIC に戻ってきます。これは A=X と同じ処理になります。

RUN させて X にいろいろな値(実数, 整数)を入れると、A の値が X の値と等しいことが分かります。それでは 50~70 行を次のように、変更して RUN して下さい。

```

50 INPUT X$
60 A$=USR(X$)
70 PRINT A$:GOTO 50

```

? TM Error in 60が表示されてしまいます。N-BASICではこれはエラーになりませんが、N₆₀-BASICでは機械語から戻ってきたときに数値型であるかチェックしており、文字型はTM Errorになるように処理されています。それゆえに文字型は使用できないからです。しかし、これもワークエリアを操作すれば、簡単に、文字型も扱えるようになります。

9-3 引数

9-3-1 数値型

引数は、FAC(浮動小数点アキュムレータ)を通して受渡されます。このFACのアドレスはFF65~FF6AHです。

N-BASICにおいては、整数型、単精度型、倍精度型の明確な区別がありましたが、N₆₀-BASICにおいては実数型しかなく、整数型は、実数型に変換されてFACに入ります。そのため、整数型を引数とした場合は実数型から整数型への変換をしなければなりません。その変換ルーチンに当たるのは、N₆₀-BASIC 活用表に例としてのっている、0741H 番地です。

逆に機械語より BASIC に受渡すときは、整数から実数型へ変換しなければなりません。それが、0D16H 番地(A, Bレジスタに値をセットすること)になります。

整数型の例として、エラーコードを引数とし、そのエラーメッセージを出力するプログラムを示します。N₆₀-BASICではエラーコードは0~40までの偶数だけですので奇数は入れないで下さい。

```

10 CLEAR 300,&HDF00-1
20 FOR I=&HDF00 TO &HDF1F:READ A$:POKE I,VAL("&
  h"+A$):NEXT I
30 POKE &HFAEB,0
40 POKE &HFAEC,&HDF
50 INPUT X
60 A=USR(X)
70 PRINT :GOTO 50
100 DATA cd,41,07,21,9b,03,19,3e,3f,cd,c7,26,7e,
  cd,c7,26
110 DATA 23,7e,cd,c7,26,21,84,03,cd,cf,30,af,32,
  25,ff,c9

```

実数型の引数はそのまま FAC に入るので FAC の内容をそのまま取り出すだけで使えます。 $2\pi/X$ の計算をさせるプログラムを載せます。

```
10 CLEAR 300,&HDF00-1
20 FOR I=&HDF00 TO &HDF0F:READ A$:POKE I,VALC("&
  h"+A$):NEXT I
30 POKE &HFAEB,0
40 POKE &HFAEC,&HDF
50 INPUT X
60 A=USR(X)
70 PRINT A:GOTO 50
100 DATA cd,41,39,21,e5,3d,11,66,ff,cd,1a,39,cd,
  06,38,c9
```

9-3-2 文字型

文字型はエラーになることは前に述べました。これは機械語からの戻り先が 0BD8H 番地になっておりここで数値型のチェックをしているからで、この 0BD8H 番地へ、戻るのが機械語の部分で変更すれば文字型を使えるようになります。

```
10 A$="ABCDEFGH"
20 C$=USR(A$)
```

上のプログラムを実行すると、FF39H に文字列の長さ、FF3BH, FF3CH に先頭アドレスがセットされます。次は、文字列を引数として渡し、その文字列を逆に CRT に出力するプログラムです。

```
10 CLEAR 300,&HDF00-1
20 FOR I=&HDF00 TO &HDF18:READ A$:POKE I,VALC("&
  h"+A$):NEXT I
30 POKE &HFAEB,0
40 POKE &HFAEC,&HDF
50 INPUT X$
60 A$=USR(X$)
70 PRINT :GOTO 50
100 DATA 3a,39,ff,2a,3b,ff,06,00,4f,09,2b,47,7e,
  cd,c7,26
110 DATA 2b,10,f9,e1,cd,fa,0a,e1,c9
```

文字型の引数を BASIC に戻す例として、ひらがな→カナ変換のプログラムを挙げます。

```
10 CLEAR 300,&HDF00-1
20 FOR I=&HDF00 TO &HDF14:READ A$:POKE I,VALC("&
  h"+A$):NEXT I
30 POKE &HFAEB,0
40 POKE &HFAEC,&HDF
50 INPUT X$
60 A$=USR(X$)
70 PRINT A$:GOTO 50
100 DATA 3a,39,ff,2a,3b,ff,47,7e,cd,4f,1a,77,23,
  10,f8,e1
110 DATA cd,fa,0a,e1,c9
```

9-4 BASICを機械語で

9-4-1 ファンクションキー・イニシャライズ

ファンクションキーのイニシャライズ(初期設定)をする場合、N-BASIC では、3A81H 番地より160バイト分ブロック転送するだけの簡単な処理で済みましたが、N₆₀-BASIC ではキー入力の章で説明したようにメモリの節約のために、初期設定の内容が中間言語+1文字の2バイトになっています。

この2バイトをファンクションキーのバッファに入れるのは、BASIC を使うより機械語の方が簡単にできます。

```
10 CLEAR 300,&HDF00-1
20 FOR I=&HDF00 TO &HDF2F:READ A$:POKE I,VALC("&
  H"+A$):NEXT I
100 DATA 06,0a,21,3d,fb,11,67,01,c5,e5,1a,d5,cd,
  65,06,e6
110 DATA 7f,77,1a,13,23,fe,00,30,f6,d1,13,1a,77,
  13,23,af
120 DATA 77,e1,01,00,00,09,c1,10,df,cd,b5,12,c9,
  00,00,00
```

使い方

EXEC &HDF00

9-4-2 KEY LIST

ファンクションキーの内容を見るプログラムは前に BASIC でつくりましたが、それを機械語に置き換えてみます。使い方は KEY LIST とします。

key	list	
F-1	COLOR	F-6 SCREEN
F-2	CLOAD"	F-7 CSAVE"
F-3	GOTO	F-8 PRINT
F-4	LIST	F-9 PLAY
F-5	RUN	F-10 CONT
Ok		

```
10 CLEAR 50,&HDF00-1
20 FOR I=&HDF00 TO &HDF6B
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT
50 POKE&HFAB1,0:POKE&HFAB2,&HDF
100 DATA FE,97,C2,53,23,D7,E5,11,65,FB,21,3D,FB,
    3E,01,CD
110 DATA 2D,DF,EB,C6,05,CD,2D,DF,F5,3E,0D,CD,C7,
    26,3E,0A
120 DATA CD,C7,26,F1,EB,D6,04,FE,06,20,E4,E1,C9,
    E5,D5,F5
130 DATA 21,69,DF,CD,CF,30,F1,F5,6F,26,00,CD,A1,
    3A,3E,20
140 DATA CD,C7,26,F1,D1,E1,F5,E5,06,08,7E,23,B7,
    28,0B,FE
150 DATA 20,30,02,3E,20,CD,C7,26,10,F0,3E,20,04,
    CD,C7,26
160 DATA 10,FB,E1,01,08,00,09,F1,C9,46,2D,00
```

9-4-3 日本語エラーメッセージ

N₆₀-BASIC のエラーメッセージは2文字の省略形で表示されるために、入門用としては使いにくくなっています。その内容を理解するまでは、エラーメッセージ一覧表を頻繁に見ることになり、非常に不便です。

そこでエラーメッセージを日本語で表示させてみましょう。

エラーが発生すると Eレジスタにエラーコードをセットして 0401H 番地に飛んできます。そしてこの番地以降のプログラムでエラーを表示しています。エラー処理ルーチンはディスク命令等の追加のためにフック番地を RAM に設けています。それでこのフック番地を利用して、エラーメッセージを変更することができます。

```
10 CLEAR 300,&HD9FF
20 FOR I=&HDA00 TO &HDA3F:READ A$:POKE I,VAL("&
h"+A$):NEXT I
30 AD=&HDA3E:FOR I=0 TO 20:READ A$,A:C=LEN(A$):
FOR J=1 TO C
40 POKE AD+J-1,ASC(MID$(A$,J,1)):NEXT J:POKE AD
+J-1,A
50 AD=AD+J:NEXT I
60 POKE&HFF8D,&HC3:POKE&HFF8E,0:POKE&HFF8F,&HDA
100 DATA cd,f9,34,d5,cd,7a,13,cd,cd,1b,d1,cd,2d,
27,cb,0b
110 DATA 1c,43,21,3d,da,af,be,23,20,fc,10,fa,e5,
2a,5d,fa
120 DATA 7c,a5,3c,28,09,cd,a1,3a,21,35,da,cd,cf,
30,e1,cd
130 DATA cf,30,c3,41,04,ca,de,dd,c6,b5,b2,c3,20,
00,00,00
200 REM J=ERROR MSG DATA
210 DATA "FORカ` ナイノ NEXTカ` アル。",0
220 DATA "ファンボウ アマリ。",0
230 DATA "GOSUBテ` ナイノ RETURNカ` アル。",0
240 DATA "DATAカ` タリマゼン。",0
250 DATA "カンズウ=スタートメントノ アタイカ` オカシイ。",0
260 DATA "ケイサン ケツカカ` オカシイ。",0
270 DATA "メモリー カ` タリマゼン。",0
280 DATA "シテイ シタ キ` ヨウバンゴウカ` アリマゼン。",0
290 DATA "ハイレツ ソエシ` ノ アタイカ` オカシイ。",0
300 DATA "ハイレツヲ 2カイ テイキ` シタ。",0
310 DATA "0テ` フリザンシタ。",0
320 DATA "INPUT, DEFFN ハ DIRECTテ` ハ `ツカエマゼン。",0
330 DATA "ヘンスウ ノ カタカ` チカ` イマス。",0
340 DATA "モブリョウイキ カ` タリマゼン。",0
350 DATA "モ` ノ ナカ` サ カ` 255ヲ コエタ。",0
360 DATA "モ` ノ シキカ` フツザ` ツ スキ` マス。",0
370 DATA "CONTフノウ。",0
380 DATA "FNハ テイキ` サレタイマゼン。",0
390 DATA "テープ` ノ ヨミマチカ` イテ` ス。",0
400 DATA "OPERANDカ` タリマゼン。",0
410 DATA "ファイル` ノ DATA` ノ カタカ` オカシイ。",0
```

第10章 ランダムテクニック

- 10-1 TIME
- 10-1-1 タイム機能
- 10-1-2 タイマのセット
- 10-2 知っていればおもしろいランダムテクニック
- 10-2-1 CLOAD PRINT
- 10-2-2 GOTO
- 10-3 アンリストの方法
- 10-4 SCREENのもう一つの使い方
- 10-5 画面を消して実行速度アップ
- 10-6 1行は71文字以上可能か?
- 10-7 拡張ROMエリアの使い方
- 10-8 PRINTとLPRINTの切り換え
- 10-9 PEEK, POKEを使って省メモリ化
- 10-10 アベンド
- 10-11 行番号を0にする方法
- 10-12 PRESETをPSETとしても使える
- 10-13 グラフィックで相対座標が使える
- 10-14 エラーの音を変えてみよう
- 10-15 SOUND, PLAY関係のデフォルト値

第10章 ランダムテクニック

10-1 TIME

10-1-1 タイム機能

PC-6001 は時計用の専用の IC を持たないため、1/512sec のタイマ割り込みをカウントアップしてそれをタイマがわりに使用しています。

```
10 CLS
20 LOCATE 8,8
30 PRINT TIME
40 GOTO 20
```

上記のプログラムを実行させると下位の桁がめまぐるしく変化しているのが分かります。これはタイマが 1/512sec (1.953ms) ごとに 2 つずつカウントアップしているためです。

PC-6001 の内部ではカウンタ用のバッファが 4 バイト設定されています。TIME 関数は、このバッファの内容を 10 進数に直します。10 進では 4 桁目がほぼ秒の単位になりますが、この機能はあくまでカウンタであって時計の機能ではありません。仮に 61 秒になっても表示は、101XXX にならずに 61XXX となります。このように時、分、秒を自動的に処理してくれませので、時計として使用する場合はプログラムで時、分、秒の処理を行なわなければなりません。

またこのカウンタ機能では 1000 カウントが 1000/1024 秒 (0.9765625 秒) になりますので、これを 1 秒とすると、ゲーム等なら問題ないでしょうが、実用とするなら以下のように補正が必要です。

$$\text{SECOND} = \text{INT}(\text{TIME} / 1024)$$

10-1-2 タイマのセット

PC-8001 のときは `TIME$="00:00:00"` でタイマを 0 にセットできましたが PC-6001 ではそれができません。ではどうすればカウンタを 0 にセットできるのでしょうか？

方法1. リセットまたは電源を OFF にして再び ON にする。

方法2. カウンタのバッファを 0 に書き換える。

上記の方法が考えられます。

方法 1 は非現実的なので方法 2 について説明します。

カウンタのバッファは FA28～FA2BH 番地を使用していますのでここを 0 に書き換えます。

```

10 PRINT "START";TIME
20 FOR I=&HFA28 TO &HFA2B:POKE I
,0:NEXT I
30 PRINT "SET ";TIME
Ok
RUN
START 8476
SET 38
Ok

```

20行を実行した後30行で表示されるまで時間がかかるので表示された時間は0になっていません。

機械語で使用するとき、

```

0000 210000   SETIME:LD     HL,0000H
0003 2228FA           LD     (0FA28H),HL
0006 222AFA           LD     (0FA2AH),HL
0009 C9               RET

```

このタイマ機能には、PC-8001より優れている部分があります。それはタイマ割り込みを止めることによってカウントアップを一時停止できるのです。これをうまく使えば累計タイムを測ることが簡単にできます。

```

10 PRINT "START",TIME
20 P=PEEK(&HFA27):P=P OR 1:POKE &HFA27,P
30 OUT &HB0,P
40 PRINT "TIMESTOP",TIME
50 FOR I=1 TO 300:NEXT
60 PRINT "COUNT UP",TIME
70 P=PEEK(&HFA27):P=P AND &HFE:POKE &HFA27,P
80 OUT &HB0,P

```

```

RUN
START          44222
TIMESTOP       44272
COUNT UP     44272
Ok

```

50行はただの時間かせぎです。50行で時間を浪費したのにもかかわらず時間は増えていません。

ポート B0H がタイマ割り込みの ON OFF に使っているので、上記のプログラムでは20行でまずポート B0H の状態(モータのリレー状態、表示しているページ等)を調べてタイマを OFF にして、ポート B0H に出力します。そして70行でタイマを ON にして B0H に出力します。

```

0000 3A27FA  STOPTM:LD  A, (0FA27H)
0003 F601      OR      1
0005 D3B0      OUT     (0B0H), A
0007 3227FA    LD      (0FA27H), A
000A C9        RET

```

: BIT 0 を 1 にする(タイマー OFF)
: ポート出力

```

0000 3A27FA  ENTMTM: LD  A, (0FA27H)
0003 E6FE      AND     0FEH
0005 D3B0      OUT     (0B0H), A
0007 3227FA    LD      (0FA27H), A
000A C9        RET

```

: BIT 0 を 0 にする(タイマー ON)

注 ポートB0H



FA27H番地・ポートB0Hのステータスエリア

※DI命令のために正確なカウンタとしては使えない。
※タイマOFFにしたままでいると入力時にカーソルが表示されず、困ることがある。

図10-1

10-2 知っていればおもしろいランダムテクニック

10-2-1 CLOAD PRINT

セーブしたプログラムのベリファイを行なうには CLOAD ? 命令を使いますが、CLOAD PRINT でも、ベリファイを行なうことができます。これは、PRINT の省略形である "?" が、中間コード変換において本来の "?" と同じものであるとみなされるために起こります。

10-2-2 GOTO

GOTO のつづりは文字と文字との間がいくら空いていても GOTO 命令と判断します。

```
10 G _ _ O _ _ _ T _ _ _ _ O 100
```

上のプログラムの LIST をとると

```
10 GOTO 100
```

文字と文字とのスペースがつめられて、正しい命令になっています。

GOTO の書き方には一般的には GOTO と GO _ T O があります。N-BASIC では GOTO と GO _ T O の2つ中間言語を持っていて処理をしていましたが N₆₀-BASIC では GOTO だけしかなく、そのかわりに、無条件に文字間のスペースを読みとばすようにして GO _ T O を処理しているために起こる現象です。

10-3 アンリストの方法

LIST を実行してもプログラムリストをとれなくする方法です。

LIST を見せないようにする理由として第一に考えられるのは、プログラムの保護でしょう。

営業用のデモプログラム等の盗用が問題なのはもちろんのこと、個人間でも、自作のプログラムを許可なくコピーされるのは、腹立たしいものです。

アンリストする方法は2つあります。

1. LIST, LLIST のジャンプテーブルを書き換えて、LIST の処理をさせないようにする。
2. 最初の行番号を 65535 (FFFFH) に書き換える。

1 の方法では、EXEC で LIST の処理ルーチンに飛ばすか、LIST のジャンプテーブルを元に戻すことによって、表示される欠点があります。

そこで2の方法について説明しましょう。

BASIC プログラムの先頭行(例のプログラムでは10行)は 8401H 番地(16K 時は C401H)より始まっていることは前の章で説明しました。このときプログラムの先頭の行番号は 8403, 04H (C403, 04H) 番地に格納されています。この部分を POKE を使って FFH に書き換えるとリストが取れなくなります。

例

LIST

```
10 REM カケザン
20 INPUT "A="; A
30 INPUT "B="; B
40 PRINT "A×B="; A*B
50 END
Ok
poke&h8403,&hff:poke&h8404,&hff
Ok
LIST
Ok
```

このアンリストにしたプログラムをSAVE しますと、次に LOAD したときに LIST が見えません。

それを調べてみましょう。

```

csave"カケザン"
Ok
new
Ok
cload"カケザン"
Found:カケザン
Ok
list

Ok
RUN
A=? 256
B=? 16
A*B= 4096
Ok

```

ただしこの方法では問題になることがあります。N₆₀-BASIC では GOTO、GOSUB 命令があると、先頭の行から飛び先のアドレスを探すために、行番号が FFFFH になっていると飛び先がないと判断して、そこで探すのをやめるので、UL Error を出力してしまうのです。

```

10 REM カケザン
20 INPUT "A=";A
30 INPUT "B=";B
40 PRINT "A*B=";A*B
50 GOTO
Ok
poke&h8403,&hff:poke&h8404,&hff
Ok
RUN
A=? 256
B=? 16
A*B= 4096
?UL Error in 50
Ok

```

これでは困りますのでプログラム中で行番号をもとに戻さなければなりません。ただ単に戻しただけでは **STOP** キーを押して LIST を見ることができますので、LIST と LLIST の飛び先も変更します。

```

10 REM カケザン
12 POKE&H8403,10:POKE&H8404,0
14 POKE&HFA8F,&HE0:POKE&HFA90,7
16 POKE&HFA91,&HE0:POKE&HFA92,7
20 INPUT "A=";A
30 INPUT "B=";B
40 PRINT "A*B=";A*B
50 GOTO 10

```

```
poke&h8403,&hff:poke&h8404,&hff
```

```
Ok
```

```
LIST
```

```
Ok
```

```
RUN
```

```
A=? 256
```

```
B=? 16
```

```
A×B= 4096
```

```
A=?
```

```
Break in 20
```

```
Ok
```

```
LIST
```

```
Ok
```

市販のゲームの一部には、もう少し高度なことを行なっているものがあります。

たとえば、プログラムの途中(REM 文の後など)の行番号を FFFFH にし、そのすぐ後に STOP キーのキャンセルを行ない、続いて LIST, LLIST の飛び先を変更する、などというものです。次のプログラムを入力し、POKE &H851A, &HFF : POKE &H851B &HFF を実行して下さい(SAVE するのを忘れずに)。

```
10 REM *****
20 REM *      SAMPLE PROGRAM      *
30 REM *              of          *
40 REM *      UNLIST              *
50 REM *      (C) 1982            *
60 REM *              by          *
70 REM *      ASCII SYSTEMSOFT   *
80 REM *      written by FUJIKEN  *
90 REM *****
100 POKE&H851A,&H64:POKE&H851B,0
110 CLEAR 50,&HDF00-1:GOSUB 300
120 POKE&HFA8F,&HE2:POKE&HFA90,7
130 POKE&HFA91,&HE2:POKE&HFA92,7
200 SCREEN 3,2,2:COLOR 2,1,2:CLS
210 RX=INT(RND(1)*5)+1:RY=INT(RND(1)*5)+1
220 PSET(128,186)
230 IF RY>RX THEN A=RY:B=RX:GOTO 250
240 A=RX:B=RY
250 C=INT(A/B):D=A-C*B:IF D<>0 THEN A=B:B=D:GOTO
    250
260 FOR I=0 TO 6.35/B STEP .1/B
270 LINE-(128*SIN(RX*I)+128,90*COS(RY*I)+96)
280 NEXT
285 IF INKEY$="" THEN 285
290 GOTO 200
300 FOR I=&HDF00 TO &HDF26:READ A$:POKE I,VAL(" &
    H"+A$):NEXT
310 EXEC&HDF00:RETURN
320 DATA 21,0D,DF,22,02,FA,21,18,DF,22,14,FA,C9,
    C5,06,00
330 DATA F5,CD,78,0E,FE,03,18,09,C5,06,01,F5,CD,
    78,0E,FE
340 DATA FA,CA,2D,0F,C3,BC,0E
```

ところで、エラーのある行の行番号を FFEH として実行させると、どのようなことが起
こるでしょうか？ 試してみると興味深いでしょう。

LIST

```
10 TECH-KNOW
Ok
poke&H8403,&Hfe: poke&H8404,&Hff
Ok
LIST
Ok
RUN
```

10-4 SCREEN のもう一つの使い方

SCREEN はディスプレイモードの設定および、画面の切り換えでしたが、それ以外に SCREEN
関数と言われる使い方がります。

N₈₀-BASIC 活用表にあるように、SCREEN 関数は SCREEN(X, Y)の形で使われ、画
面上の座標(X, Y)の位置にあるキャラクタを調べます。

LIST

```
10 LOCATE 10,10:PRINT CHR$(65)
20 PRINT SCREEN (10,10)
Ok
RUN
```

```
65      A
Ok
```

プログラム例を見れば分かるように“A”の文字を画面に書いた後 SCREEN 関数で呼びだ
すと、“A”のキャラクタ・コードである65(41H)が表示されます。この SCREEN 関数で求
められる値は、キャラクタ・コード表と同じ値になります。たとえば“月”であったとすると
値は1になります。

```
10 CLS:AD=&H8200:FOR I=0 TO 255
20 POKE AD+I,I:NEXT
30 FOR Y=0 TO 7:FOR X=0 TO 31
40 LOCATE 0,10:PRINT "X=";X;"      Y=";Y;"      ";SCR
   EEN (X,Y):"
50 NEXT X,Y
```

このプログラムを実行すると全キャラクタを表示し、SCREEN 関数でその値を表示します。

1行は71文字までですが、これはあくまでキーバッファが71文字分しか確保されないため、そのため、普通は LIST をとると 1行が71文字以内になってしまいます。ところが、ある特定の場合のみ LIST 時で71文字以上出力されることがあります。

```
10 PRINT:PRINT:PRINT:PRINT:PRINT
PRINT:PRINT:PRINT:PRINT:PRINT:P
RINT:PRINT:PRINT:PRINT:PRINT:PRI
NT:PRINT:PRINT:PRINT:PRINT:PRINT
PRINT:PRINT:PRINT:PRINT:PRINT:P
RINT:PRINT:PRINT:PRINT:PRINT:PRI
NT:PRINT"1"
Ok
```

反対に、キー入力は71文字以内であるのに、LIST をとるとその途中までしか受け付けられていない場合もあります。

71文字キー入力したのに、LISTをとると40文字しか表示されておらず、後ろがすべてカットされています。これはグラフィックキャラクタが14H+キャラクタコードの2バイトで表わすために、キーバッファにはグラフィックキャラクタを1文字入力することに2バイトず

つ使用して起こる現象です。グラフィックキャラクタを1行に多用するときは、文字の欠落に注意する必要があるでしょう。

10-7 拡張 ROM エリアの使い方

4000~7FFFH 番地は拡張 ROM エリアになっており、ここにユーザーがつくったプログラムを ROM に入れて使用することが可能です。また、電源 ON とともに拡張 ROM のプログラムを走らせることもできます。

4000H 番地に 41H("A"), 4001H 番地に 42H("B")が書き込んであると 4002, 03H 番地に書かれているアドレスをコールします。

例

4000H	41H	} 5000番地を CALL する
1	42H	
2	00H	
3	50H	

N₆₀-BASIC では初期設定、画面クリア、ファンクションキーの表示を終了したときに、この 4000, 01H 番地を調べます。そして"A", "B"が書いてあれば次の処理先である 0084H ("How Many Pages?"を出力する部分)をスタックに入れます。

機械語だけのプログラムならば、問題はありませんが、BASIC+機械語、あるいは内部ルーチンを多用しているプログラムではページ数の設定を行なう必要がある場合もあります。

この部分は N₆₀-BASIC の内部を理解していませんと使いこなすことができません。

10-8 PRINT と LPRINT の切り換え

画面出力のプログラムをそのまま使ってプリンタに出力したいとき、あるいは PRINT を LPRINT に書き換えたいときがあります。LIST を表示して PRINT の前に "L" を 1 つずつ付け加えるのも 1 つの方法ですが、これではあまりにも幼稚過ぎます。これ以外の方法をいくつか挙げてみましょう。

1. プログラムをつくる時点であらかじめ PRINT を使わずに PRINT#-X にしておいて変数 X の値を 0 または 3 にして CRT とプリンタを切り換える。

2. CRT とプリンタの切り換えのフラグをあらかじめセットしておいて、インタプリタ内の一文字出力処理で振り分ける (PC-8001 の PRINT, LPRINT の切り換えはこの方法を用いているのが多い)。

3. RAM 上に命令のジャンプテーブルがあるのを利用して POKE 文で直接 PRINT と LPRINT の飛び先を入れ替える。

このような方法があります。2 の方法ですと若干ですが機械語のプログラムが必要です。

ここでは 3 の方法について説明します。LPRINT は 087AH 番地、PRINT は 087EH 番地にその処理プログラムがあります。また、PRINT のジャンプテーブルは FA8B, 8CH になっています。この番地には 087EH 番地がセットされています。これを 087AH 番地に書き換えますと PRINT 命令でありながら LPRINT 命令の働きをします。

```
10 PRINT "1—CRT 2—プリンタ"
20 AS=INKEY$: IF AS="" THEN 20
30 IF AS="1" THEN GOSUB 200:GOTO 100
40 IF AS="2" THEN GOSUB 300:GOTO 100
50 GOTO 20
100 PRINT "0123456789":GOSUB 200:GOTO 10
200 REM PRINT
210 POKE &HFA8B,&H7E:POKE &HFA8C,8
220 RETURN
300 REM LPRINT
310 POKE &HFA8B,&H7A:POKE &HFA8C,8
320 RETURN
```

GOSUB 200 で CRT に、GOSUB 300 でプリンタに切り換わります。

10-9 PEEK, POKE を使って省メモリ化

メモリを直接読み書きする PEEK, POKE 命令はゲームで使う以外に実用ソフトで、おもしろいことができます。

たとえば成績処理プログラムにおいて生徒の点数を配列に入れるのが普通の使い方ですが、これですと1教科について5バイトずつメモリを使用します。テストの点数は0～100までで表わされます。そうすると点数に関しては5バイトの実数型を使う必要はないわけです。1バイトで0～255まで扱える PEEK, POKE を使用することによって5倍のデータを扱うことができます。

次に2次元の配列を PEEK, POKE でつくった場合の例を示します。

```
10 XMAX=100:YMAX=10:AD=&HD000
10000 DA=PEEK(XMAX*Y+X+AD):RETURN
11000 POKE XMAX*Y+X+AD,DA:RETURN
```

XMAX	}	添字の最大値
YMAX		
AD		ストアするメモリの先頭アドレス
X,Y		配列の添字
GOSUB10000		メモリよりデータを読み出す。
GOSUB11000		メモリにデータをしまう。

10-10 アペンド

複数のプログラムをアペンド(結合)したい場合があります。方法としては2とおりあります。まず1つめはただ単にプログラム④の後ろにプログラム⑤を付け加えるもので、2番目の方法はPC-8001ではMERGEと呼ばれる方法でプログラム④と⑤とを結合させ、もし同じ行番号があれば後からLOADした⑤の内容が残ります。また、④と⑤に同一の行番号がない場合には重ね合わせる形で結合します。

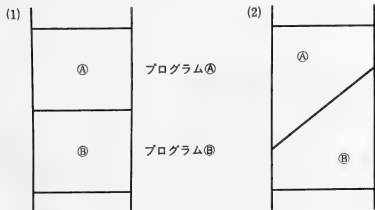


図10-2 アペンド状態

N₈₀-BASICには行番号を変更する命令がありませんので、このような完全なアペンド機能をつくることはできません。

それでは1の方法について説明します。

プログラムA

```
10 PRINT "PROGRAM A"  
20 FOR I=1 TO 10 :NEXT
```

プログラムB

```
100 PRINT "PROGRAM B"  
110 GOTO 10
```

プログラム④に⑤を、結合する場合、すでにプログラム⑤はカセットにSAVEされているものとします。

1. まずプログラム④を入力(キー入力またはLOAD)します。

LIST

```

10 PRINT "PROGRAM A"
20 FOR I=1 TO 10:NEXT _____1
Ok
a=peek(&hff56)+peek(&hff57)*256- _____2
2
Ok
poke&hfa5f,a-int(a/256)*256:poke _____3
&hfa60,int(a/256)
Ok

```

2.FF56,57H番地(変数領域開始番地)の内容を読み出して、プログラム④をキー入力した場合は2、カセットからロードした場合には9引く。

3.2で求めた値を FA5F,60H(プログラム開始番地)に下位, 上位の順にセットする。

ここで LIST をとると何も表示されないはずですが、これはプログラム開始アドレスをプログラム④の終了アドレスに変更しているためです。

```

LIST
Ok
cload
Found: B _____4
Ok
LIST
100 PRINT "PROGRAM B"
110 GOTO 10
Ok

```

4.プログラム⑥を LOAD します。LIST をとりますと、プログラム⑥が表示されます。

```

poke&hfa5f,1:poke&hfa60,&h84 _____5
Ok
LIST
10 PRINT "PROGRAM A"
20 FOR I=1 TO 10:NEXT
100 PRINT "PROGRAM B"
110 GOTO 10
Ok

```

5.FA5F,60H(プログラム開始アドレス)を元に戻します。LIST をとりますと、④と⑥が結合されています。

注) RAM が 16KB の場合は次のようにして下さい。

```

poke&hfa5f,1:poke&hfa60,&hc4
Ok

```

10-11 行番号を0にする方法

時々、マイコン雑誌に発表されたプログラムや市販されているゲームプログラムで、一部行番号を0にしてあるのを見かけます。

```
0 REM *****
0 REM * PC-6001 *
0 REM * Graphic Mahjong *
0 REM * Ver1.3 *
0 REM * program *
0 REM * by Ray Kazuto *
0 REM * sound & title design *
0 REM * by Yellow Panther *
0 REM *****
```

このようにする深い意味はないのですが、0 REM(ゼロレムと呼んでいます)にすると、なんとなくプログラムが高級に見えます。

このプログラムは BASIC で簡単につくることができます。

行番号を0にするプログラム

```
65000 INPUT "No. ";NO:CU=PEEK(&HFA5F)+PEEK(&HFA6B)*256
65010 LN=PEEK(CU+2)+PEEK(CU+3)*256
56
65020 IF NO<LN THEN END
65030 POKE CU+2,0:POKE CU+3,0
65040 CU=PEEK(CU)+PEEK(CU+1)*256
:GOTO 65010
```

上記のプログラムを行番号を0にしたいプログラムの後に付け加えます。そして RUN 65000 とすると "No." と表示されますから、変更したい所までの行番号を入れます。しばらくすると "Ok" と表示されますので、LIST をとると行番号が0になっています。

例

```
10 REM *****
20 REM * PC-6001 *
30 REM * Graphic Mahjong *
40 REM * Ver1.3 *
50 REM * program *
60 REM * by Ray Kazuto *
70 REM * sound & title design *
80 REM * by Yellow Panther *
90 REM *****
65000 INPUT "No. ";NO:CU=PEEK(&HFA5F)+PEEK(&HFA6B)*
256
65010 LN=PEEK(CU+2)+PEEK(CU+3)*256
65020 IF NO<LN THEN END
```

```
65030 POKE CU+2,0:POKE CU+3,0
65040 CU=PEEK(CU)+PEEK(CU+1)*256:GOTO 65010
```

100行までを0にする場合

```
run 65000
No.? 100
Ok
LIST
```

```
0 REM *****
0 REM * PC-6001 *
0 REM * Graphic Mahjong *
0 REM * Ver1.3 *
0 REM * program *
0 REM * by Ray Kazuto *
0 REM * sound & title design *
0 REM * by Yellow Panther *
0 REM *****
```

10-12 PRESET をPSET としても使える

PSET ではX、Y座標とカラーを指定しますが PRESET でもカラーの指定ができます。

PRESET(10, 10), 2

エラーにならずに点を表示します。これは PSET(10, 10), 2 と同じ機能になっています。

N₆₀-BASIC インタプリタでは、PRESET の処理は Acc に背景の色をセットし、PSET の処理ルーチンに飛ばしているため、カラー指定ができるわけです。

10-13 グラフィックで相対座標が使える

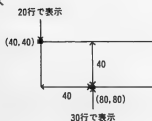
グラフィック命令では、絶対座標を指定して、点や線を表示しますが、STEP を使うことによって、相対座標で使うこともできます。

この機能は1つ前の座標を原点として、STEP で指定した値を加算した値を次の座標とします。

```
10 SCREEN 2,1,1:CLS
20 PSET(40,40),1
30 PSET STEP(40,40),2
```

このプログラムを実行してみてください。点が2個表示されます。

Ok



```
1 COLOR CLOAD GOTO LIST RUN
```

20行で表示された点の位置から(40, 40)加算された位置に点が表示されています。この相対座標の指定には当然のことながらマイナスの数も使えます。

```
PSET STEP(-40,-40),4
```

この相対座標の指定は PSET の他、PRESET, LINE, PAINT, POINT 命令でも使うことが可能です。

例 LINE

```
10 SCREEN 2,1,1:CLS
20 LINE(80,40)-STEP(40,40),2
```

Ok



1 COLOR CLOAD GOTO LIST RUN

```
10 SCREEN 3,2,2 :CLS
20 PSET(0,0),2
30 FOR I=1 TO 10
40 LINE STEP(6,3)-STEP(4,2),2
50 NEXT
```



10-14 エラーの音を変えてみよう

サウンド機能のちょっとおもしろい利用法です。

N₆₀-BASIC のエラー処理ルーチンはワークエリア内に 2 ケ所フックを持っています。これを利用して遊んでみましょう。

エラーを出すと“ピッ”という音が出ますが、これではあまりに単調なので、エラーを出すとヒューという音がして、音楽まで流れます。(いささかうるさいという気もしますが…)。

次のプログラムを間違いなく入力して RUN します。OK が表示されたら、もうこのプログラムは不要ですから NEW を実行してかまいません。リセットボタンを押すまでは、エラーを出すたびにしつこく音を出します。リセットボタンを押してしまったときは、プログラムの最後の 4 行の POKE 文をダイレクトで実行してください。また復活します。

電源を切ってしまったら仕方ありません。また同じプログラムを入力してください。

また、このプログラムは、機械語を使用していますから、とくにデータの部分は間違いなく入力してください。また危険防止のため入力が終わったら、必ずカセットに SAVE してから RUN させましょう。

```
10 REM*****
20 REM*   NEW ERROR SOUND   *
30 REM*       for PC-6001   *
40 REM*       by YELLOW PANTHER *
50 REM*****
60 CLS:PRINT "Now writing machine language";
70 FOR ADDRESS=&HDF00TO&HDF7A
80 READ DA$:POKE ADDRESS,VAL("&h"+DA$):PRINT". "
;
90 NEXT ADDRESS
110 DATAf5,c5,d5,cd,b3,1b,3e,00
120 DATA5f,cd,c5,1b,3e,01,1e,00
130 DATAcd,c5,1b,7b,1e,78,cd,c5
140 DATA1b,06,06,c5,06,0a,1d,cd
150 DATAc5,1b,c5,06,ff,10,fe,c1
160 DATA10,f4,06,14,1c,cd,c5,1b
170 DATAc5,96,ff,10,fe,c1,10,f4
180 DATAc1,10,e0,cd,b3,1b,21,4f
190 DATAdf,cd,b3,1e,3a,1b,fd,b7
200 DATA20,fa,d1,c1,f1,c9,00,22
210 DATA54,32,30,30,4f,53,31,33
220 DATA4d,31,30,30,4c,38,44,34
230 DATA44,2e,44,31,36,44,34,46
240 DATA2e,45,31,36,45,2e,44,31
250 DATA36,44,2e,43,2b,31,36,44
260 DATA34,22,00
270 REM HOOK INITIALIZE
290 POKE&HFF8D,&HCD
300 POKE&HFF8E,&H00
310 POKE&HFF8F,&HDF
320 POKE&HFF90,&HC9
330 PRINT "Complete!"
```


10-15 SOUND, PLAY 関係のデフォルト値

PC-6001 の電源投入時の PSG の値は次のようになっています。

レジスタ番号	データ
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	56
8	0
9	0
10	0
11	0
12	0
13	0

本によってはレジスタ 8, 9, 10 のデータが違う値になっているものもありますが、これは 0 が正解のようです。

また PLAY 命令の MML のそれぞれのデフォルト値は次のようになっています。

MML	データ
V	8
L	4
Ö	4
S	1
M	255

参考にしてください。



付録

- 付-1 I/Oポート一覧表
- 付-2 N60-BASICインタプリタ一覧表
- 付-3 ワークエリア一覧表
- 付-4 中間言語と処理ルーチン対応表
- 付-5 キャラクタ・コード表
- 付-6 キーボード配列表
- 付-7 エラーメッセージ一覧表
- 付-8 タイマー・モニタ
- 付-9 12平均率音階表
- 付-10 サウンドレジスタ一覧表

付-1 I/O ポート一覧表

ポートアドレス		内 容																												
10進	16進																													
128	80H	(入出力)μPD8251(USART)データワード (入出力)μPD8251(USART)コントロールワード } RS-232C用 (NEC μPD8251データシート参照のこと)																												
129	81H																													
130	82H	80H, 81Hのイメージ																												
131	83H																													
143	8FH																													
144	90H	(入出力)ポート A μPD8049とのコミュニケーションに使用 (モード2で使用)																												
145	91H	(出 力)ポート B プリンタデータ(モード0で使用)																												
146	92H	(入出力)ポート C																												
		<table><tr><td>ビット</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>入出力</td><td>入力</td><td>出力</td><td>入力</td><td>出力</td><td>入力</td><td>出力</td><td>出力</td><td>出力</td></tr><tr><td>内 容</td><td>8049</td><td>8049</td><td>8049</td><td>8049</td><td>8049</td><td>CG 切り換え</td><td>CRT KILL</td><td>プリン タスト ローブ</td></tr></table> モード2		ビット	7	6	5	4	3	2	1	0	入出力	入力	出力	入力	出力	入力	出力	出力	出力	内 容	8049	8049	8049	8049	8049	CG 切り換え	CRT KILL	プリン タスト ローブ
ビット	7	6	5	4	3	2	1	0																						
入出力	入力	出力	入力	出力	入力	出力	出力	出力																						
内 容	8049	8049	8049	8049	8049	CG 切り換え	CRT KILL	プリン タスト ローブ																						

付-2 N₈₀-BASIC インタプリター一覧表

〔用語解説〕

DW アセンブラの疑似命令でワード定数を示すもの(DEFW)

FAC 浮動小数点アキュムレータのこと。FF66~FF6AH までの FAC を FAC①、FF6D~FF71H までの FAC を FAC②と呼ぶことにしました。

〔マーク解説〕

○	サブルーチンマーク	ユーザーが簡単に使えるようなサブルーチンを示す。
□	予約語マーク語	<p>N₈₀-BASIC の予約語の処理ルーチンを示す。 このマークが付いているルーチンは下記のような使い方ができます。</p> <pre>PRINT"ABC"</pre> <p>内部 95H , 22H, 41H, 42H, 43H, 22H, 00H このアドレスを HL に入れて { RST 10H CALL 087EH (PRINT の処理アドレス) で、PRINT "ABC" が実行されます。(第2章を参照)</p> <p style="text-align: right;">(中間言語形式)</p>
☒	予約語マーク2	N ₈₀ -BASIC の予約語のうち、□印で示したような使い方ができないもの、無意味なものにこの☒をつけている。
△	ルーチン・マーク	サブルーチンではないが、ユーザーが使えるようなアドレスを示す。

〔注意〕

- コメント文で示した例(ex)の中には、使用方法を誤ると暴走するものもありますから、コメント文の内容だけで理解できる方のみ使って下さい。
- コメント中の A はアキュムレータ、B、C、D、E、H、L はレジスタ、SP はスタックポインタのことです。
Z=ゼロフラグ CY=キャリーフラグ

マーク	アドレス	内 容	コ メ ン ト
	0 0 0 0	コールド スタート	ハード・ソフトのイニシャライズ後0040Hへ行く。
	0 0 0 3	テーブル (DW 0741H)	FAC①の値を DE に入れるルーチンのアドレスが示されている。
	0 0 0 5	テーブル (DW 0D16H)	A, Bレジの値を FAC①に入れるルーチンのアドレスが示されている。
○	0 0 0 8	パラメータ チェック	<p>(HL)と RST 8H 命令の次のアドレスのデータとを比較し、違えば SN Error となる。 パラメータが合えば一文字解析ルーチンへ行き、比較データの次のアドレスへ戻る。 ex) 09EB : RST 8H 09EC : DB 3BH ; ' '(比較データ) 09ED : PUSH HL ← (HL)と 3BH を比較し、合えば、09EDH へ戻る。</p>
○	0 0 1 0	1文字解析	HL を1つ進めた後(HL)の内容を Aに入れる。Aの値が、3AH(コロン)か 00Hならば Z=1, 数字を表す ASCIIコード(30 ~ 39H)ならば CY=1として戻る。
	0 0 1 8	RST 18H 用フック (JP FFDBH)	RST 18H を実行すると FFDBH 番地へ飛ぶ、初期設定では FFDBH 番地には RET 命令(C9H)があるので、何もしないが、ユーザーがこれを変更して使うことができる。
	0 0 1 B	テーブル (DW FAEBH)	USR 関数の処理アドレスが格納されているアドレスが示されている。
○	0 0 2 0	HL とDE の比較	<p>HL>DE のとき Z=0, CY=0 HL=DE のとき Z=1, CY=0 HL<DE のとき Z=0, CY=1 となる。また HL, DE の内容は変化しないが、A は変化する。</p>
	0 0 2 6	テーブル (DW FA1FH)	CMT のボーレートを示すフラグのアドレスを示している。 (第4章を参照)
○	0 0 2 8	符号 チェック	<p>FAC①の数の符号を調べる。 負の時 A=FFH Z=0 0の時 A=00H Z=1 正の時 A=01H Z=0 となる。</p>

0 0 3 0	RST 30H 用フック (JP 4030H)	RST 30Hは拡張ROMへ飛ぶようになっているため、ユーザーがそのまま使うことはできない。
0 0 3 8	RST 38H 用フック (JP FFE1H)	0018H と同様
0 0 4 0	イニシャライズ 2	N ₈₀ -BASIC ソフトのイニシャライズ・ルーチンの始まり。
0 0 4 3	ファンクションキー初期設定	第 4 章参照
0 0 6 F	拡張 ROM チェック	第10章参照
0 0 8 4	ページ数設定	How Many Pages? の入力とページ関係データの設定
0 0 D 6	初期メッセージ表示	N ₈₀ -BASIC ↓ Bytes free の表示を行なう。 表示終了後 BASIC テキストのエディット (0442H) へ行く。
0 0 F 0	メッセージ データ	Bytes free CR/LF
0 0 F E	データ	ページ数によるフリーアドレスの上位バイトの値 F9H : ページ 1 の時 DFH : ページ 2 の時 BFH : ページ 3 の時 9FH : ページ 4 の時
0 1 0 2	メッセージ データ	How Many Pages?
0 1 1 1	メッセージ データ	N ₈₀ -BASIC.....1981 CR/LF
0 1 3 4 ↓ 0 1 9 4	データ	ワークエリアの初期データ FA00~FA60H まで。
0 1 9 5 ↓ 0 1 E A	ジャンプテーブル (コマンド)	各命令の処理アドレスが示されている。これらのデータは、ワークエリア内に移される。 第 2 章の 6 項を参照。
0 1 E B ↓ 0 2 1 C	ジャンプテーブル (関数)	

	0 2 1 D ↓ 0 2 D 4	中間言語名テーブル (コマンド)	<p>キーワードを ASCII 英大文字で格納してある。ただし、それぞれキーワードの最初の 1 文字だけマスク(ビット7を 1 にしている)してある。</p> <p>PC-8001 では中間コードも一緒にテーブル内に持っていたが PC-6001 では、最初から何番目に出てきた中間言語かカウントし、それをもとに計算して中間コードを求めている。</p> <p>ex) C5, 4E, 44 } END 80H C6, 4F, 52 } FOR 81H</p>
	0 2 D 5 ↓ 0 3 6 D	中間言語名テーブル (関数)	
	0 3 6 E	中間言語名テーブルの終了マーク	
	0 3 6 F ↓ 0 3 8 3	演算用テーブル	<p>+, -, *, /, ^, AND, OR の順に、その優先度と処理アドレスが示されている。</p> <p>ex) 036FH : 79, 7E, 36</p> <p>↑の処理アドレスは 367EH で優先度を示す。</p> <p>データは 79H である。</p>
	0 3 8 4	メッセージデータ	Error
	0 3 8 B	メッセージデータ	in
	0 3 9 0	メッセージデータ	Ok CR/LF
	0 3 9 5	メッセージデータ	Break
	0 3 9 B ↓ 0 3 C 4	エラーメッセージデータ	<p>エラー番号順に2バイトづつデータが並ぶ</p> <p>ex) 039BH : 4E, 46 } NF エラー 039DH : 53, 4E } SN エラー</p>
	0 3 C 5	FOR, NEXT RETURN 用サブルーチン	
	0 3 E 4	エラー処理	READ 処理中、データのタイプミスマッチによるエラーの処理ルーチン
	0 3 E A	SN エラー	<p>以下このアドレスへジャンプするとエラーが表示され、エラー処理を行なう。</p> <p>ex) EXEC &H03FF</p> <p>とすれば TMエラーとなる。</p>
	0 3 E D	/0 エラー	
	0 3 F 0	NF エラー	
	0 3 F 3	DD エラー	
	0 3 F 6	UF エラー	
	0 3 F 9	OV エラー	
	0 3 F C	MO エラー	
	0 3 F F	TM エラー	
	0 4 0 1	エラー処理	Eにエラー番号を入れてここへジャンプすると、そのエラーの名前を表示し、エラー処理を行なう。

△	0 4 4 1	EDIT2	POP BC 後 EDIT へ
△	0 4 4 2	EDIT (ホットスタート)	BASIC のテキストエディット (コマンド待ち)
	0 4 7 8	BASIC テキスト操作	行の変更追加, 削除を行なう
	0 4 8 F		行の削除
	0 4 A 3		1 行分あける
	0 4 A E		1 行挿入
	0 4 C 4		BASIC テキストのポインタ変更 (第 2 章参照)
○	0 4 E 5	LIST sub	LIST のサブルーチン LIST 行番号を求める DE に開始行番号, (SP) に終了行番号を入れ, 開始行番号の行サーチを行ない戻る。
	0 5 0 1	行サーチ	DE に行番号を入れ, コールすると, i) $\left\{ \begin{array}{l} Z=0 \\ CY=0 \end{array} \right\}$ 指定行はない, BC に次の行 HL に次の次の行のアドレス ii) $\left\{ \begin{array}{l} Z=1 \\ CY=1 \end{array} \right\}$ 指定行があり, BC にその行 HL に次の行のアドレス iii) $\left\{ \begin{array}{l} Z=1 \\ CY=0 \end{array} \right\}$ プログラムのどの行よりも 指定行が大きい のいずれかで戻る。
○	0 5 1 E	中間言語変換ルーチン	FEDAH からのバッファの ASCII コードに よる BASIC テキストを中間言語に直して FEDAH からのバッファへ入れる。 HL に FEDAH を入れて戻る。
□	0 5 D 6	LLIST	I/O 出力先をプリンタにして LIST へ
□	0 5 D B	LIST	
○	0 6 6 5	中間言語をキーワードにな おす	A に中間コードを入れてコールすると, A に キーワードの頭文字, DE にその命令のキ ーワードテーブルでのアドレスを入れ, 戻る。
☒	0 6 7 E	FOR	
	0 6 E A	BASIC 実行メインルーチ ン	マルチステートメントおよび行終了のチェッ クを行ない, 行番号をワークエリアに入れて, 各命令の処理アドレスへ飛ぶ。
○	0 7 2 A	1 文字解析 2	HL を 1 つ進めてから, HL の内容が 20H (ス ペース) でなくなるまで HL を進め, 1 文字解 析 (0016H) と同様の処理を行なう。

<input type="radio"/>	0739	整数入力	HL を1つ進めて、そのアドレスの内容から式解析を行ない整数値を DE に取りこむ。エラーチェックもある。
<input type="radio"/>	073A	"	073AHからのルーチンはHLを1つ進めないだけ ex) PRINT&H1000 内部 95, 26, 48, 31, 30, 30, 30 ↑ このアドレスを、HL に入れてコールすると、DE に 1000Hが入る。
	0755	FC エラー	ここへジャンプすれば FC エラーになる。
	075A ↓ 075E	データ	数値データ -32768 を示す。
<input type="radio"/>	075F	行番号入力	HL 以降の10進 ASCII 数字列を入力して DE に入れる。 これは、行番号の入力に使われているルーチンのため FFFAH 以上は入力されない。
<input type="checkbox"/>	0781	RUN	
<input checked="" type="checkbox"/>	078F	GOSUB	
<input type="checkbox"/>	07A0	GOTO	
<input checked="" type="checkbox"/>	07BC	RETURN	
<input checked="" type="checkbox"/>	07E0	DATA	C=3AH REM. DATA 共有ルーチン参照
<input checked="" type="checkbox"/>	07E2	REM	C=00H REM. DATA 共有ルーチン参照
<input type="radio"/>	07E4	REM. DATA 共有ルーチン	HL の内容が、C または 00H と一致するまで HL を進める。ただしダブルクォーテーション内でのデータは00H以外無視する。 ex) DATAはコロン(3AH)または00HまでHLを進める REMは00Hまで HLを進める。
<input type="checkbox"/>	07F5	LET	
	080E	文字変数代入	
	083D	数値変数代入	
<input checked="" type="checkbox"/>	0844	ON	
<input checked="" type="checkbox"/>	0861	IF	
<input type="checkbox"/>	087A	LPRINT	I/O出力先をプリンタにしてPRINT
<input type="checkbox"/>	087E	PRINT	PRINT#のチェックも行なっている。

	0 9 0 2	" , " 処理	
	0 9 2 6	SPC, TAB 処理	
	0 9 5 0	" ; " 処理	
	0 9 5 5	PRINT#, INPUT#終了処理	
○	0 9 6 7	-255~0 数値入力	HLを1つ進めてそのポインタより数値を入力し、Aに入れる、値が自然数なら FC エラーへ行く。
	0 9 7 8	メッセージデータ	? Redo from start CR/LF
	0 9 8 B	INPUT, READ共通サブルーチン	主にエラーの判定を行なう。
☒	0 9 A B	INPUT	
	0 9 C 4	RS-232C 指定 INPUT	
	0 9 C F	CMT 指定 INPUT	
	0 9 E 0	キーボード指定 INPUT	
☒	0 A 0 9	READ	
	0 A 0 E	INPUT, READ 共有ルーチン	(FF49H)=0 なら INPUT, (FF49H)≠0 なら READ として処理
	0 A C 5	メッセージデータ	? Extra ignored CR/LF
	0 A D 6	READ 用サブルーチン	
○	0 A F 6	数式解析	HL からの式の解析を行ない、それが数値型でなければ TM エラーとなる。 データは FAC①に入っている。
○	0 A F 9	数値型チェック	数値型でなければ TM エラー
○	0 A F A	文字型チェック	文字型でなければ TM エラー
○	0 B 0 4	式解析 2	(HL)が D2H('='の中間コード)であることを確認してから式解析へ
○	0 B 0 7	式解析 3	(HL)が 28H('('の ASCIIコード)であることを確認してから式解析へ このルーチンは主にカッコ内の式解析に使われる。
○	0 B 0 9	式解析	HL からの式の解析を行ない、データを FAC ①に格納する。

○	0 B 6 6	>, =, < 処理 1	
	0 B 8 1	式解析サブルーチン	条件にあった処理アドレスへジャンプさせる。
	0 B C C	- 符号処理	
	0 B D D	変数処理	
	0 B E E	英小文字→英大文字	HL の内容が、英小文字なら英大文字にして A に入れる。
	0 B F 8	&H 16進数入力	& で始まれば、&H 以下 16 進数を入力し、そうでなければ 075FH (行番号入力) へ行く。DE に入力したデータが入る。
○	0 C 3 B	関数処理	拡張用フック有り、式解析サブから A = (中間コード-D4H) でコール
	0 C 9 9	OR 処理	
	0 C 9 A	AND 処理	
	0 C B F	>, =, < 処理 2	
	0 C D 1	テーブル (DW 0CD3H)	>, =, < 処理 2 の続きが、0CD3H であることを示している。
	0 C D 3	>, =, < 処理 3	
	0 C F 9	NOT 処理	
	0 D 1 0	FAC①←(HL-DE)	HL-DE を行ない、その結果を FAC①に格納する HL, DE, BC, A 変化
	0 D 1 5	整数(A, C)を FAC①に格納	整数の上位1バイトを A に、下位1バイトを C に入れてここをコールすると FAC①にデータが格納される。(レジスタの値は確保されないで注意)
	0 D 2 2	LPOS	
○	0 D 2 7	POS	
	0 D 2 B	A の値を FAC①に格納(整数 0~255)	A にデータを入れてここをコールすると FAC①にデータが格納される。
	0 D 3 0	CSRLIN	

<input checked="" type="checkbox"/>	0 D 3 A	DEF	<p>単変数領域内に下図の形で FN 関数が定義される。</p> <p>10 DEF FNLM(X)=X+1</p> <p>FN関数名 LM FN関数ポインタ X 引数名 X ダミー</p> <p>FN関数は数値変数として扱われるため、最後の1バイトがダミーとしてある。(第2章参照)</p>
<input checked="" type="checkbox"/>	0 D 6 1	FN	
<input type="checkbox"/>	0 D A F	ダイレクト処理チェック	ダイレクトモードならば ID エラー。A 以外のレジスタは変化しない。
	0 D B D	FN 関数名解析	
<input checked="" type="checkbox"/>	0 D C C	INP	
<input type="checkbox"/>	0 D D 6	OUT	
<input type="checkbox"/>	0 D E 3	1 バイト整数入力	<p>HL を 1 つ進めて式解析後、結果を A に入れる。ただしその値が、0~255まででなければ、FC エラーとなる。Eレジにも A と同じものが入っている。</p> <p>ex) OUT 255, 10</p> <p>内部 90, 32, 35, 35, 2C, 31, 30, 00</p> <p>↑ (*)</p> <p>このアドレスを HL に入れてコールすると A には、FFH (10進数の255)が入って戻る。</p>
<input type="checkbox"/>	0 D E 4	1 バイト整数入力 2	最初に HL を 1 つ進めない以外 0DE3H と同じ。例で言えば(*)のアドレスを HL に入れてコールすれば良い。
<input checked="" type="checkbox"/>	0 D F 3	PEEK	
<input type="checkbox"/>	0 D F A	POKE	
<input type="checkbox"/>	0 E 0 6	2 バイト整数入力	<p>式解析後 DE に整数値が入る。</p> <p>ex) POKE &HA000, 0</p> <p>内部 94, 26, 48, 41, 30, 30, 30, 2C, 30, 00</p> <p>↑</p> <p>このアドレスを HL に入れてコールすると DE に A000H が入る。</p>
	0 E 1 0	PC-6001のイニシャライズ	ワークエリアの設定、ハードのイニシャライズ後 0040Hへ行く

○	0 E 7 8	サブ CPU ハンドシェイク 1文字入力	A に8049からのデータが入る。
○	0 E 8 F	サブ CPU ハンドシェイク 一文字出力	A のデータを8049へ送る。
	0 E A 8	キー割り込み処理①	CMT使用時にSTOPキーを押すと、このルーチンへくる。データは強制的に03H である。
	0 E B 0	キー割り込み処理②	グラフィックキー、ファンクションキー、STOPキーを押すと、この処理ルーチンへくる。サブ CPU から受取るデータは、グラフィックキーならその ASCII コード、ファンクションキーなら F0~F9H、STOP キーなら FAH となっている。
	0 E B 5	キー割り込み処理③	一般のキーを押すとこのルーチンへくる。サブ CPU から受け取るデータは、押したキーの ASCII コードである。
	0 E B 8	キー関係割り込みの共有 ルーチン	クリック音の発生、STOP・ESC フラグの設定、バッファへのデータ出力。 グラフィックキー・ファンクションキーに14H のマークをつけるのもこのルーチン
	0 F 3 1	STOP キー割り込みsub	STOP キーを押すと(CMT 使用時はダメ)このルーチンへきて、PLAY を停止させる。 最初にフック(CALL FFD8H)があるのでSTOP キーのキャンセルができる。 ex) FFD8H : JP STOP STOP : POP AF LD A, 07H (STOPキーでベルがなる) JP 0EF6H
	0 F 4 0	ゲーム用キー割り込み { STICK } { STRIG }	{ 1061H STICK, STRIG データ入力ルーチンを参照 }
	0 F 4 A	RS-232C用割り込み	データをバッファへためる バッファフルなら受信を一時中止する。
	0 F 7 4	2 msec タイマ割り込み	TIME のカウント カースルの点滅 PLAY の発生を行なう。
	0 F 9 F	CMT READ 割り込み	FA1DH にデータをセットしデータがきたことを示すフラグをたてる。 (FA19H の bit 1 を 1 にする)
	0 F B 2	割り込み	8049 からの割り込みではないらしい。(未使用)

	0FB7	CMT エラー割り込み	エラーを示すフラグをたてる。 (FA19H の bit 4 を 1 にする)
○	0FBC	1 文字入力 sub (キーボードから) グラフィック文字は 1 文字 入力の説明を参照	入力待ちがない。キー入力があれば Z=1 で 戻り、キー入力があれば A にデータを入れ Z=0 で戻る。 このルーチンは、リアルタイムの 1 文字入力 に使えるが、キーバッファが働いていること に要注意。 また、すべてのモードで使用可能。一文字入 力ルーチンのように入力後 TEXT モードに なることはない。 (A 以外のレジスタの値は保存される)
○	0FC4	1.文字入力 (キーボードから) グラフィック文字は 14H とそれに続けて ASC II コードに 30H をプラスし た値で示されるため 2 回入 力しなければならない。 ex) 画は 14H, 31H である。	入力があるまで、カーソルを点滅させて待つ。 入力があれば、カーソルの点滅を止め、A に データを入れて戻る。 一文字入力 sub でも述べたが、入力がある と、TEXT モードになるので、グラフィック モード設定時にこのルーチンを使うときは要 注意。 ページ切り換えキーの操作、ファンクション キーの表示等もこのルーチン内で操作してい る。
○	0FFF	1 文字入力 sub 2 グラフィック文字は 1 文字 入力の説明を参照	1 文字入力 sub と同じだが、レジスタの値が 確保されないから、ユーザーは 1 文字入力 sub の方を使った方がよい。
○	103A	キーバッファより 1 文字入 力	キーバッファより A に 1 文字取ってくる。た まっているデータがなければ Z=1 で戻り データがあれば Z=0 で A にデータを入れ戻 る。
○	1041	ファンクションキーデータ 読み込みルーチン	ファンクションキーカウンタ(FA32H)が、 セットしてある場合ファンクションキーの データを順に取り出して Z=0, A にデータを 入れ戻る。 カウンタがセットされていない、もしくは データが 0 のとき Z=1 で戻る。
○	1058	キーバッファクリア	キーバッファをクリアする。 ここをコールするだけでバッファに入ってい るデータがなくなるのでゲーム等のプログラ ムで使えば有用

○	1061	STICK・STRIG データ入力 (キーボード用)	<p>サブCPU にコマンド 6 を送り STICK・STRIG 関係のキー入力状況を調べさせる。サブCPU は調査が済むと割り込みをかける。メインCPU は割り込み処理ルーチンでデータを受取り (FECAH) にセットする。(FECAH) のデータの内容は下図のとおりで、キーが押されたビットが 1 になる。</p> <div style="text-align: center;"> bit 7 6 5 4 3 2 1 0 (FECAH) = <table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>スペース</td> <td>常に</td> <td>←</td> <td>→</td> <td>↑</td> <td>↓</td> <td>ストップ</td> <td>シフト</td> </tr> </table> </div> <p>このルーチンをコールするとデータを A に入れて戻る。</p>	スペース	常に	←	→	↑	↓	ストップ	シフト
スペース	常に	←	→	↑	↓	ストップ	シフト				
○	1075	CRT 1 文字出力 グラフィック文字は 14H に 続けて ASCII コードに 30H をプラスした値を送る と表示される。	<p>A にデータを入れコールすると、カーソル位置より表示する。14H のときはグラフィックフラグをたてるだけで次にこのルーチンをコールしたときグラフィック文字としての補正を行なう。</p> <p>コントロールコードは、07H (CTRL-G), 0AH (CTRL-J), 0BH (CTRL-K), 0CH (CTRL-L), 0DH (CTRL-M), 1CH (→), 1DH (←), 1EH (↑), 1FH (↓) の処理のみ行なわれる。</p>								
	108B	14H (グラフィック文字) チェック									
○	10AA	CRT 表示 1	A に ASCII コードを入れコールするとカーソルの位置から表示する。								
○	10D9	CTRL-G	BEEP 音を発生する。								
	10DE	PC-6001 ソフトの初期設定	拡張 RAM のチェック、ワークエリアの初期設定等を行なう。								
○	116A	カーソル実アドレスセット	FDA8H, A9H に示された X+1, Y+1 のカーソル位置の、モード 1 における V-RAM 上のアドレスを求め、FDAA, ABH にセットする。								

○	1 1 6 D	カーソルロケーション	H: X+1, L: Y+1を入れコールするとその位置にカーソルを移動する。1文字表示などを行なえば、指定位置にカーソルが移動したことがはっきりする。 X, Y の値の確認は行なわれないので要注意。
○	1 1 7 9	カーソル SW ON	カーソル表示のフラグをたてる。 全レジスタの値が確保される。
○	1 1 8 1	カーソル SW OFF	カーソル表示のフラグをクリアして反転の状態なら元に戻す。
○	1 1 9 1	カーソル反転	テキスト、またはセミグラフィックの場合のみカーソル反転を行なう。
○	1 1 B 8	カーソル左端実アドレス算出	Lに Y+1を入れコールすると、モード1における画面左端の V-RAM のアドレスを DE に入れ戻る。DE 以外のレジスタは不変
○	1 1 C D	カーソル実アドレス算出	Hに X+1, Lに Y+1を入れコールすると、モード1におけるその位置の V-RAM のアドレスを HL に入れ戻る。HL 以外のレジスタは不変
○	1 1 D A	1行消去ルーチン	Lに Y+1, DEにその左端の TEXT アドレス (CALL 11B8H で求められる) を入れコールすると SCREEN の第2パラメータに応じたページの Y 行を消す。グラフィックモードでも可能 ex) SCREEN 1, 1, 1のとき <pre>LD L, 01H CALL 11B8H CALL 11DAH RET</pre> を実行させると 0 行が消去される。
	1 2 1 7	モード3用1行消去ルーチン	
	1 2 1 F	モード4用1行消去ルーチン	
○	1 2 6 0	スクロールアップ	Hにスクロール最上行+1, Lにスクロール最下行+1を入れコールすると、画面のスクロールアップが行なわれる。全モードで可能 また、Aレジのみ変化
	1 2 6 8	スクロールアップ、ダウン共有ルーチン	Z=0のときスクロールアップ処理、Z=1のときスクロールダウン処理が行なわれる。

○	1 2 A 9	スクロールダウン H:スクロール最下行+1 L:スクロール最上行+1	使い方は、スクロールアップと同じ、 ex) SCREEN 1, 1, 1 にて LD HL, 1001H CALL 12A9H RET を実行すると画面が1段下がる。
○	1 2 B 5	ファンクションキー表示	画面下段のページ数とファンクションキーの表示を行なうのがこのルーチン。
○	1 3 6 8	キャラクタ反転	COLOR の第1パラを操作して、キャラクタを反転させる。次に表示されるものが反転される。もう一度このルーチンをコールすればもとに戻る。
○	1 3 7 A	SCREENをTEXT モードにする。	グラフィック(モード3,4)のときページ1のTEXT モードに SCREEN を設定し直す。
○	1 3 8 A	SCREEN モードチェック	モード1の時 Z=0 CY=1 モード2の時 Z=1 CY=1 モード3,4の時 Z=0 CY=0 で戻る。
○	1 3 9 0	SCREEN 第1パラ設定	Aに(第1パラメータ)-1の値を入れコールするとアクティブページのモードを変更する。
○	1 3 DB	アトリビュートデータ算出	Aに(SCREEN 第1パラ)-1の値を入れコールすると COLOR 第3パラを考慮に入れたアトリビュートのデータをAに入れ戻る。
	1 3 E 9 ↓ 1 3 E C	アトリビュート基本データ	順にモード1, 2, 3, 4のアトリビュートの基本データが入っている。 ex) 13E9H: 20H モード1のアトリビュートデータ
○	1 3 E D	SCREEN 第3パラ設定	Aに(第3パラメータ)-1の値を入れコールすると表示ページが切り換わる。
○	1 4 0 C	SCREEN 第2パラ設定	Aに(第2パラメータ)-1の値を入れコールすると、アクティブページが変更される。
	1 4 1 F sub	SCREEN 第2パラ設定	アクティブページデータをそのページのワークに退避して、新しいページのデータをアクティブページデータエリアに入れる。
	1 4 2 A	アクティブページ・データをページ・データエリアへ移す。	SCREEN 第2パラで指定されたページへ移す。

	1 4 3 3	SCREEN 第1パラ設定 sub	モード変更に伴う COLOR データの受け渡しを行なう。COLOR データを現在のモードの COLOR データ領域へ移し、指定モードの COLOR データをアクティブ COLOR データ領域へセットする。
	1 4 4 9	現在の COLOR データをモードにあった COLOR データ領域へ移す。	
	1 4 5 2	各モード COLOR と各ページデータ領域のアドレス算出ルーチン	
○	1 4 7 8	カーソル実アドレスをグラフィック実アドレスに変換	HL にカーソル実アドレスを入れコールすると、HL にグラフィックモードでの V-RAM のアドレスが入り戻る。 HL 以外のレジスタは不変
○	1 4 A 0	グラフィックモード用キャラジェネアドレス計算	A に ASCII コードを入れコールすると、CGROM でのそのコードのデータの開始アドレスを DE に入れ戻る。DE 以外のレジスタは不変
○	1 4 A F	CRT 表示 2	A に ASCII コード HL にカーソルの実アドレスを入れコールすると、その位置に1文字表示を行なう。カーソルの移動は行なわない。
	1 4 C 9	CRT 表示 sub	
	1 4 D 5	テキスト、セミグラモード処理	
	1 4 E 2	グラフィックモード処理	
	1 4 F F	カラーグラフィック sub	
	1 5 2 E	フルグラフィック sub	
	1 5 4 0	スーパーインポーズ	
	1 5 4 9	スクロールアップダウン sub	

1578 15BA 15C0 15C8 15D1 15D5 15D9 15DD 15E1 15E5	COLOR コマンド0 COLOR コマンド1 COLOR コマンド9 COLOR コマンド2 COLOR コマンド3 COLOR コマンド4 COLOR コマンド5 COLOR コマンド6 COLOR コマンド7 COLOR コマンド8	COLOR 関係のサブルーチン群 COLORコマンドジャンプテーブル参照 COLOR コマンド0, 1については不明、また詳細なことについては、まだ解析されていないコマンドもある。
15E9 ↓ 196E	各 COLOR コマンドの 処理ルーチン	COLOR コマンドジャンプテーブル
196F ↓ 19BE	COLOR コマンド ジャンプテーブル	2 バイトずつテキスト用、セミグラフィック用 カラーグラフィック用、フルグラフィック用の 順で各コマンド処理アドレスが示されている。
196F ↓ 19BE	COLOR コマンド ジャンプテーブル	ex) 196F: E9, 15 コマンド0 テキスト用 15E9H 1971: FB, 15 コマンド0 セミグラ用 15FBH 1973: 09, 16 コマンド0 カラーグラ用 1609H 1975: 0D, 16 コマンド0 フルグラ用 160DH 1977: 0F, 16 コマンド1 テキスト用 160FH :

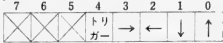
コード \ コマンド	0	1	2	3	4	5	6	7	8	9
テキスト	15E9	160F	168F	16AD	1725	17B3	1800	1869	1889	1668
セミグラフィック	15FB	161F	1695	16D5	1744	17D5	181C	187D	18A5	1678
カラーグラフィック	1609	1647	168F	16AD	172F	17BB	1808	1585	18B0	167E
フルグラフィック	160D	1658	169D	16AD	173B	17CD	1814	1885	18B0	168A

コマンドNo.	内 容
0	不明
1	不明
2	COLOR 第1パラメータの範囲を調べ、それを越える数について補正を行なう。
3	各COLOR に対応したアトリビュートの値を求める。
4	現在のアトリビュートの COLOR コードを求める。
5	現在の位置より1つ後のアトリビュートでのアドレスを求める。
6	現在の位置より1つ前のアトリビュートでのアドレスを求める。
7	現在の位置より1つ上のアトリビュートでのアドレスを求める。
8	現在の位置より1つ下のアトリビュートでのアドレスを求める。
9	アトリビュートデータのセット(グラフィックモードのときはその色のビットパターンをセット)

注: コマンド7, 8にはまだ不明なルーチンも含まれているので内容が異なる可能性もある。

○	1 9 B F	RS-232C イニシャライズ	RS-232C 用バッファのクリア, 8251のイニシャライズを行なう。
○	1 9 D 1	RS-232C 1文字入力	1文字入力するまで待つ, キーボードの1文字入力と同様に, バッファから1文字読んでくるのであって, リアルタイムなものではない。 STOP, ESC キーのチェックを行なっているので無限ループに入ることはないだろう。
○	1 9 E 8	RS-232C 1文字出力	A の内容を1文字出力
○	1 9 F D	8251 イニシャライズ	
○	1 A 0 F	LCOPY sub	サーマルプリンタ用グラフィックデータ1バイト出力ルーチン。 Cにデータを入れコールする。ただしプリンタの方はあらかじめグラフィック出力にしておかなければならない。B, C, Aが変化する。
○	1 A 1 C	プリンタ1文字出力	Aにデータを入れコールする。グラフィック文字は無視され, ひらがなはカタカナに変換されて印字される。
	1 A 2 B	プリンタ1文字出力 sub	Aにデータが入り, Cに1が入っていると, BUSY のとき STOP, ESC キーが, チェックされる。Cが1でないときは, STOP, ESC キーは, きかなくなる。
○	1 A 4 F	ひらがな→カタカナ変換	Aにデータを入れコールするとひらがなのASCIIコードならカタカナのコードに変換して戻る。
○	1 A 6 1	CMT ロード用イニシャライズ	PC-6001 では, SAVE 時と LOAD 時でのイニシャライズ, ストップが同一ではないので注意して欲しい。このルーチンでは8049にCMT(LOAD用)をOPENすることを宣言しモーターをONにする。
○	1 A 7 0	CMT 1文字入力	Aにデータが入る。Z=0のときは, テープリードエラーでZ=1の時はLOADに成功している。
○	1 A A A	CMT LOAD 用 STOP	8049にCMT(LOAD用)をCLOSEすることを宣言し, モーターをOFFにする。
○	1 A B 8	CMT SAVE 用イニシャライズ	モーターをONにして8049にCMT(SAVE用)をOPENする。
○	1 A C C	CMT 1文字出力	Aにデータを入れコールすると出力される。
	1 A E D	CMT SAVE の READY チェック	ここをコールして, Z=0で戻れば準備OK。









○	1 B 0 6	CMT SAVE用 STOP	8049にCMT(SAVE 用)を CLOSE することを宣言し、モーターを OFF にする。
	1 B 1 4	CMT 関係コマンドを8049へ送る。	600ボー SAVE OPEN 3DH, 39H 600ボー LOAD OPEN 1DH, 19H 1200ボー SAVE OPEN 3EH, 39H 1200ボー LOAD OPEN 1EH, 19H の2バイトを8049へ送る。
○	1 B 2 A	*のプリnk表示	CLOAD のとき(*)をプリnkさせる。1 回コールすると、(*)を表示していれば消し、表示していなければ(*)を表示する。
	1 B 4 0	CMT 割り込み用 ワークの関係ビットのクリア	(FA19H)の bit 1 と bit 4 を 0 にする。 0F9FH, 0FB7Hを参照
○	1 B 4 9	モーター OFF	
○	1 B 4 B	モーター ON	
	1 B 5 1	2msecタイマ割り込みON	
○	1 B 5 4	システムポート (B0H)出力	B に変更するビットだけ1にしたものを入れ A に新しい出力用データを入れコールする。 (FA27H)に出力したデータが格納される。
○	1 B 6 0	PLAY STOP sub	PLAY用ワークとバッファのクリア, PSG の音を止める。ユーザーはここをコールするよりも 1BB3H をコールした方が良い。
	1 B B 3	PLAY STOP	PLAY 発生を止める。
	1 B B 9 ! 1 B B D	PLAY 用 ワーク の 初期データ、詳細は不明	
○	1 B B E	8910(PSG)コントロール 1	A にレジスタ No, E にデータを入れコールすると、D に変更前の指定レジスタのデータが入り、PSG に E のデータがセットされる。
○	1 B C 5	8910(PSG)コントロール 2	A にレジスタ No, E にデータを入れコールすると PSG にセットする。
○	1 B C D	BEEP 発生	CTRL-G 用のルーチン
	1 B E C	PLAY 発生ルーチン	2 msec タイマ割り込みで使用 PLAY で作られたデータを、実際に音にして発生させるルーチン。

○	1 C A 6	ジョイスティック入力	A にジョイスティックナンバー (01H か 02H) を入れコールするとジョイスティックの状態を A に入れ戻る。A 以外のレジスタは不変 <div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">bit</div> <div style="border: 1px solid black; padding: 2px; display: flex; gap: 5px;"> <div style="border: 1px solid black; padding: 2px; text-align: center;">7</div> <div style="border: 1px solid black; padding: 2px; text-align: center;">6</div> <div style="border: 1px solid black; padding: 2px; text-align: center;">5</div> <div style="border: 1px solid black; padding: 2px; text-align: center;">4</div> <div style="border: 1px solid black; padding: 2px; text-align: center;">3</div> <div style="border: 1px solid black; padding: 2px; text-align: center;">2</div> <div style="border: 1px solid black; padding: 2px; text-align: center;">1</div> <div style="border: 1px solid black; padding: 2px; text-align: center;">0</div> </div> </div> <p>A = </p> <p>ビットが 1 のときそのボタンは押されている。</p>
□	1 C D 2	LOCATE	
□	1 C F 6	CONSOLE	
○	1 D 6 B	1 バイト整数入力 3	1 バイト整数入力 2 (0DE4H) と同じだが D, E, B, C のレジスタの値は変化しない。
	1 D 7 3	CONSOLE 命令の実行	H に CONSOLE 開始行+1, L に最終行+1 が入ってコールされている。(H ≤ L)
□	1 D 9 B	COLOR	
	1 D B B	COLOR 第3パラメータ実行	A に 1, 2 を入れコールすると前回と異なるデータならば、画面全体を反転させる。
□	1 D F 8	CLS	
○	1 D F B	CLS 2	ここをコールすれば CLS が行なわれる。 ex) EXEC &H1DFB
□	1 E 0 4	SCREEN	
	1 E 5 8	ページ数チェック付1バイト整数入力	A に (データ) - 1 が入り最大ページ数と比較してそれより大きければ FC エラーとなる。
☒	1 E 6 5	SCREEN 関数処理	
☒	1 E 8 3	TIME	
□	1 E 9 B	SOUND	
□	1 E B 3	PLAY	
	2 0 0 C	テーブル (DW 200EH)	PLAY 関係処理ルーチンのジャンプテーブルの先頭アドレスを示す。
	2 0 0 E 1 2 0 3 B	PLAY 処理ジャンプテーブル	PLAY の " " 内での音階、特別の意味を持つ記号等の飛び先を3バイトで示している。音階は ASCII コード+アドレス、その他のコードは ASCII コードにマスクをかけたもの+アドレスで示される。 ex) 200E: 41, 1A, 21 音階"A"の処理は 211AH から 2023: CD, 7B, 20 "M"の処理は 207BH から

	2 0 3 C ↓ 2 0 4 A	音階用データテーブル	詳細は不明
	2 0 4 B ↓ 2 0 6 2	音階データ	C, C#, D, D#, E, F, F#, G, G#, A, A#, B の順で音階周波数の基本データが入っている。
	2 0 6 3	V(音量)処理	
	2 0 7 B	M(エンベロープ周期)処理	
	2 0 9 B	S(エンベロープ形状)処理	
	2 0 A 5	L(長さ)処理	
	2 0 B F	T(テンポ)処理	
	2 0 C C	O(オクターブ)処理	
	2 0 D 9	R(休符)処理	
	2 0 F E	N(数音階)処理	
	2 1 A A	A~G(音階)処理	
	2 2 3 1 ↓ 2 2 3 5	データ 詳細は不明	
	<input checked="" type="checkbox"/> 2 2 3 6	STICK	
	2 2 4 C	STICKキーボード処理	
<input checked="" type="checkbox"/>	2 2 5 6	STRIG	
	2 2 6 E	STRIGキーボード処理	
	2 2 7 6	STICK, STRIG 機器 No 入力	
	2 2 8 6	STICK 方向データ ジョイスティック用	
	2 2 9 6	STICK 方向データ キーボード用	
<input type="checkbox"/>	2 2 A 6	LCOPY	

	2 3 4 A ↓ 2 3 4 E	LCOPY データ I	サーマルプリンタ用グラフィックモード指定 20H スペース 20H スペース 0AH ラインフィード 1DH グラフィック指定 } 256×193のグラフィック指定 C1H 193												
	2 3 4 F ↓ 2 3 5 2	LCOPY データ II	セミグラ用のデータ 下図参照 <table><tr><td>0</td><td>0</td><td>→ 0 0 H</td></tr><tr><td>0</td><td>1</td><td>→ 0 F H</td></tr><tr><td>1</td><td>0</td><td>→ F 0 H</td></tr><tr><td>1</td><td>1</td><td>→ F F H</td></tr></table>	0	0	→ 0 0 H	0	1	→ 0 F H	1	0	→ F 0 H	1	1	→ F F H
0	0	→ 0 0 H													
0	1	→ 0 F H													
1	0	→ F 0 H													
1	1	→ F F H													
<input type="checkbox"/>	2 3 5 3	KEY													
	2 3 9 3	PLAY 用各処理ルーチン へのジャンプ処理													
<input type="checkbox"/>	2 4 7 E	CSAVE													
<input type="checkbox"/>	2 4 9 6	CLOAD													
	2 4 E C	LOAD 失敗													
	2 5 0 B	メッセージデータ	Bad CR/LF												
	2 5 1 1	CLOAD 用 file name の 格納ルーチン													
	2 5 1 8	CSAVE 用 file name の 格納ルーチン													
	2 5 3 9	CMT より file name を 格納するルーチン													
○	2 5 4 E	CMT マークチェック	CMTよりEのデータをD回検出するまで ループ												
○	2 5 6 5	file name の比較	FECBH から6文字と(HL)から6文字を比較 して合えば Z=1, 違えば Z=0, 例外として (FECBH)=0なら Z=1となる。												
	2 5 7 6	メッセージデータ	Found :												
	2 5 7 D	メッセージデータ	Skip :												
	2 5 8 3	file name 表示													
	2 5 9 A	INPUT:-1用 CMT OPEN													
	2 5 A 8	PRINT:-1用 CMT OPEN													

○	2 5 B 7	file name のCMT 出力	
	2 5 D 0	SAVE メイン部	HLにBASICの開始アドレスを入れてコールする。
	2 5 E 5	TIME ディレイ	
	2 5 F 6	LOAD メイン部	A=FFH のとき LOAD A=00H のとき VERIFY となる。 00H が 10コ検出されると終了で、Z=0で失敗、Z=1で成功である。
○	2 6 1 3	2バイト整数(アドレス)入力	HL の示すアドレスより式解析を行ない、正整数のチェックをしてから、DE にその値を入れて戻る。
□	2 6 1 D	EXEC	
○	2 6 2 7	バッファへ1文字送り込む。	AにバッファNo, Eにデータを入れコールすると、指定ナンバーのバッファへそのデータを入れる。その際バッファフルならば Z=1で戻る。
○	2 6 4 2	バッファから1文字読み出す。	A にバッファNo を入れコールすると A にデータを入れ戻る。Z=1のときは、バッファにデータがたまっていない場合。
○	2 6 6 F	バッファマップクリア	<div style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;"> A にバッファNo B にバッファの最大容量 3FH DE にバッファの先頭アドレス </div> を入れコールするとバッファをクリアする。
○	2 6 A 2	バッファの残りバイト数を求める。	A にバッファNo を入れコールすると HL に残りバイト数を入れ戻る。A にも同じ値が入っている。
○	2 6 B 1	バッファマップよりデータを読み出す。	A にバッファNo を入れコールすると、バッファの CからBまで使用していることを示し、HL が、バッファマップの3バイト目を示す。
○	2 6 B B	バッファマップアドレス算出	A にバッファのナンバーを入れコールするとバッファマップの先頭アドレスを HL に入れ戻る。
○	2 6 C 7	各I/O1文字出力	(FA58H)にI/O機器を下図のように指定し、 A にデータを入れコールすると指定I/O機器に1文字出力する。 (FA58H) = $\begin{cases} 00H & \text{CRT} \\ 01H & \text{プリンタ} \\ 02H & \text{RS-232C} \\ 80H & \text{CMT} \end{cases}$

	2 6 D A	プリンタ1文字出力	
	2 7 0 1	プリンタの CLOSE	
	2 7 1 E	CRT 1文字出力	
	2 7 2 2	CMT "	
	2 7 2 6	RS-232C "	
	2 7 2 A	1文字入力	
○	2 7 2 D	各I/OへCR/LF出力	A 以外のレジスタは不変
○	2 7 4 D	STOP・ESCキー入力 チェック	A 以外不変
☒	2 7 7 1	INKEY\$	
	2 7 8 E ↓ 2 7 9 D	CRT 1 文字出力ルーチン 用 CTRL コード処理 アドレステーブル	CTRL-J, K, L, M     の順で2バ イトずつアドレスが並ぶ。
	2 7 9 E	 処理	
	2 7 B 0	CTRL-M 処理	
	2 7 B 5	CTRL-J 処理	
	2 7 C 9	 処理	
	2 7 D 9	 処理	
	2 7 E F	CTRL-K 処理	
	2 7 F A	 処理	
	2 8 7 4	CTRL-L 処理	
	2 8 B A	SCREEN EDIT 用行フラ グのスクロールアップ	A にスクロール行数, Lに Y+1を入れコー ル。
	2 8 C D	SCREEN EDIT 用行フラ グのスクロールダウン	
	2 8 E 0	SCREEN EDIT 用行フラ グ入力	Lに Y+1を入れコールするとその行の行フ ラグを A に入れ戻る。
	2 8 E D	SCREEN EDIT 用行フラ グセット	Lに Y+1, A に行フラグデータを入れコー ルすると, そのデータを行フラグとしてセッ トする。

○	2 8 F 9	LINE INPUT	コールすると[RET]もしくは[STOP]を押すまで SCREEN EDITを行ない、HLに(入力テキストのバッファアドレス)-1を入れ戻 る。 STOP を押した時は、CY=1で戻る。																																												
	2 9 0 5	INPUT 用入力ルーチン																																													
○	2 9 6 6	コントロールコードサーチ	HLにテーブルの先頭アドレス、Cにデータ数、Aにデータを入れコールすると、Aのデータが、テーブル内にあればZ=1、なければM=1で戻る。																																												
	2 9 6 E	コントロールコード処理																																													
	2 9 C 2 ↓ 2 9 C D	コントロールコード・データ 1 (INS フラグクリア用)	1バイトずつ CTRL コードを示す。 ex) 29C2H : 0D CTRL-M(CR)を表す。																																												
	2 9 C E ↓ 2 9 D 7	コントロールコード・データ 2 (CTRL 処理用)	1バイトずつ CTRL コードを示し、これらは次のジャンプテーブルと逆の順番で対応している。																																												
	2 9 D 8 ↓ 2 9 E B	コントロールコード・データ 2 用ジャンプテーブル	<table><tr><th>データ</th><th>CTRL</th><th>アドレス</th><th>内 容</th></tr><tr><td>09</td><td>I</td><td>2 A9 2</td><td>TAB 8 カラム</td></tr><tr><td>0A</td><td>J</td><td>2 9 E C</td><td>LF</td></tr><tr><td>08</td><td>H</td><td>2 B1 2</td><td>DEL</td></tr><tr><td>12</td><td>R</td><td>2 A C 3</td><td>INS</td></tr><tr><td>02</td><td>B</td><td>2 B A A</td><td>カーソル左</td></tr><tr><td>06</td><td>F</td><td>2 B 8 6</td><td>カーソル右</td></tr><tr><td>05</td><td>E</td><td>2 B 6 C</td><td>カーソルより後を消す</td></tr><tr><td>03</td><td>C</td><td>2 A 8 2</td><td>STOP</td></tr><tr><td>0D</td><td>M</td><td>2 A 1 C</td><td>CR</td></tr><tr><td>15</td><td>U</td><td>2 B 6 4</td><td>1 行抹消</td></tr></table>	データ	CTRL	アドレス	内 容	09	I	2 A9 2	TAB 8 カラム	0A	J	2 9 E C	LF	08	H	2 B1 2	DEL	12	R	2 A C 3	INS	02	B	2 B A A	カーソル左	06	F	2 B 8 6	カーソル右	05	E	2 B 6 C	カーソルより後を消す	03	C	2 A 8 2	STOP	0D	M	2 A 1 C	CR	15	U	2 B 6 4	1 行抹消
	データ	CTRL	アドレス	内 容																																											
	09	I	2 A9 2	TAB 8 カラム																																											
	0A	J	2 9 E C	LF																																											
	08	H	2 B1 2	DEL																																											
	12	R	2 A C 3	INS																																											
	02	B	2 B A A	カーソル左																																											
	06	F	2 B 8 6	カーソル右																																											
05	E	2 B 6 C	カーソルより後を消す																																												
03	C	2 A 8 2	STOP																																												
0D	M	2 A 1 C	CR																																												
15	U	2 B 6 4	1 行抹消																																												
2 9 E C	CTRL-J																																														
2 A 1 C	CTRL-M																																														
2 A 8 2	CTRL-C																																														
2 A 9 2	CTRL-I																																														
2 A C 3	CTRL-R																																														
2 A C C	INS中なら1文字分あけるルーチン																																														
2 A D B	1文字あけるルーチン																																														
2 B 1 2	CTRL-H																																														

○	2 B 6 4	CTRL-U	
	2 B 6 C	CTRL-E	
	2 B 8 6	CTRL-F	
	2 B A A	CTRL-B	
	2 C A 9	英数カナ文字チェック	A の内容が、0~9, A~Z, a~z, カナの ASC II コードなら CY=0 ちがえば CY=1 となり戻る。
	2 C E E	グラフィック関係命令の座標入力ルーチン	STEP 処理有り
□	2 D 3 7	PRESET	
□	2 D 3 C	PSET	
	2 D 3 F	PSET. PRESET 共通ルーチン	
☒	2 D 5 5	POINT	
□	2 D C 7	LINE	
□	2 E D C	PAINT	
☒	3 0 5 B	STR\$	
	3 0 6 B	文字列と文字列データのセット	
	3 0 9 1	"(ダブルクォーテーション)の処理	
○	3 0 C F	メッセージ出力	HL にメッセージの先頭アドレスを入れコールすると、(HL)に 00H がでてくるまで表示する。
○	3 0 E 7	文字列追加	A の数だけ文字列領域に追加
	3 1 0 2	ガベージ・コレクション	
○	3 1 F 3	(BC) → (DE) L 個	BC から L バイト DE からへ移す。
○	3 1 F C	文字数チェック	式解析後にコールすると(HL)が文字数を示す。
☒	3 2 2 9	LEN	
☒	3 2 3 8	ASC	
☒	3 2 4 9	CHRS	

<input checked="" type="checkbox"/>	3 2 5 7	LEFT\$	
<input checked="" type="checkbox"/>	3 2 8 6	RIGHT\$	
<input checked="" type="checkbox"/>	3 2 8 F	MID\$	
<input checked="" type="checkbox"/>	3 2 B A	VAL	
<input checked="" type="checkbox"/>	3 2 D E	FRE	
<input type="checkbox"/> <input type="radio"/>	3 2 F D	DIM sub	DIM A(10), B(20), などのように宣言する配列が2つ以上のとき ", " のチェックを行なって DIM のルーチンへ
	3 3 0 2	DIM	
	3 3 0 7	変数解析 (DIM と共有ルーチン)	HL の示すポインタより1つの変数の解析を行ない、DE にその変数のポインタを入れ戻す。つまり DE は数値変数なら FAC①の数値データの先頭、文字変数なら文字変数データの先頭を示す。
	3 3 4 4	FN チェック	
	3 3 5 5	変数名サーチ	
	3 3 7 8	新変数登録	
	3 3 B A	配列サーチ	
	3 3 E 6	配列サーチ2	
	3 4 1 3	新配列登録	
<input type="radio"/>	3 4 A 2	メモリチェックとブロック転送	
<input type="radio"/>	3 4 B 0	メモリチェック1	C×2バイトだけの SP 領域の余裕があるか、なければOM エラーとなる。
<input type="radio"/>	3 4 B 9	メモリチェック2	通常のメモリチェックルーチン
<input type="checkbox"/>	3 4 C D	NEW	
<input type="checkbox"/>	3 5 1 9	RESTORE	
<input type="checkbox"/>	3 5 3 3	STOP	
<input type="checkbox"/>	3 5 3 5	END	
<input type="checkbox"/>	3 5 6 B	CONT	
<input type="radio"/>	3 5 A 1	英大文字チェック	CY=0で英大文字

<input type="checkbox"/>	3 5 A 9	CLEAR							
<input type="radio"/>	3 5 F 0	DE ← HL-DE	HL は変化せず						
<input checked="" type="checkbox"/>	3 5 F 7	NEXT							
<input type="radio"/>	3 6 7 E	+ 処理							
	3 6 8 3	- 処理							
	3 6 8 C	FAC① ← FAC①+FAC② (+, - 共有ルーチン)							
	3 7 4 7	4 バイト倍長整数加算	FAC①, FAC②の仮数部 4 バイトを使って FAC① ← FAC①+FAC②を行なう。						
	3 7 5 9	4 バイト倍長整数減算	加算と同様で FAC① ← FAC①-FAC②を行なう。						
	3 7 6 B	FAC①の補数	FAC① ← FAC①の補数						
	3 7 A A	5 バイト 2 倍化	HL の示すアドレスから 5 バイトの内容を 2 倍にして戻す。						
	3 7 B 4	* 処理							
	3 7 E E ↓ 3 7 F 2	数値データ 0.1							
	3 7 F 3 ↓ 3 7 F 7	数値データ 10							
	3 7 F 8	1/10化		FAC① ← FAC①×0.1を行なう					
	3 8 0 3	/ 処理							
	3 8 8 E	RST 28H の続き							
<input checked="" type="checkbox"/>	3 8 9 8	SGN							
<input checked="" type="checkbox"/>	3 8 B 9	ABS							
<input type="radio"/>	3 8 C 3	FAC①の SP 格納	<p>FAC①の内容を下図のように、スタックに退避する。</p> <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">SP ←</div> <table style="border-collapse: collapse; text-align: center;"> <tr><td>(FF65H)</td></tr> <tr><td>(FF66H)</td></tr> <tr><td>(FF67H)</td></tr> <tr><td>(FF68H)</td></tr> <tr><td>(FF69H)</td></tr> <tr><td>(FF6AH)</td></tr> </table> <div style="margin-left: 10px;">} デミー</div> <div style="margin-left: 10px;">} FAC①</div> </div>	(FF65H)	(FF66H)	(FF67H)	(FF68H)	(FF69H)	(FF6AH)
(FF65H)									
(FF66H)									
(FF67H)									
(FF68H)									
(FF69H)									
(FF6AH)									

○	38D5	FAC① → FAC②してから (SP) → FAC①	FAC①の内容を FAC②へ転送後、前記のよう に格納された FAC の内容を FAC①へ 持ってくる。
○	38EA	(SP) → FAC②	前記のように格納された FAC の内容を FAC②へ持ってくる。
	38FC	整数型データ → FAC①	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">B └── 指 数</div> <div style="text-align: center;">C 上 位</div> <div style="text-align: center;">D</div> <div style="text-align: center;">E 下 位</div> </div> <div style="text-align: center; margin-top: 5px;">└──────────┘ 仮数</div>
	3907	FAC① → 整数型データ	
○	3913	(DE) → (HL) へ4バイト転 送	
○	3917	(HL) → FAC② 5 バイト 転送	
○	391B	(DE) → (HL) 5 バイト転 送	
○	3939	FAC② → FAC① 5 バイ ト転送	
○	3941	FAC① → FAC② 5 バイ ト転送	
○	3944	FAC① → (HL) 5 バイト 転送	
○	3981	FAC①より整数型の正数 を HL に入れる。(エラー チェック有り)	
○	399B	HL を FAC①に正整数と して格納	
☒	39E7	INT	
	3A16	・ の処理	
○	3A99	in 行番号表示	in に続けて HL の値を10進表示
○	3AA1	行番号表示	HL の値を10進変換して表示
	3B70 ┆ 3B7E	数値データ1 0.5 10000,0000 1E+09	各データについて5バイト使用

	3 B 7 F └ 3 B A 2	数値データ 2 1×10 ⁸ 1×10 ⁷ 1×10 ⁶ 1×10 ⁵ 1×10 ⁴ 1×10 ³ 1×10 ² 1×10 1	各データについて 4 バイト使用
	3 B A 3	RND	
	3 C A F	FAC① → FAC②	
	3 C B 7	FAC② → FAC①	
	3 D 0 3	FAC② → (SP)	
	3 D 0 8	FAC① → (SP)	
	3 D 1 8	(SP) → FAC②	
	3 D 1 D	(SP) → FAC①	
○	3 D 2 D	FAC①の符号チェック	FAC①=0 Z=1 A=0 FAC①>0 Z=0 A=1 FAC①<0 Z=0 A=FF
	3 D 4 A └ 3 E 2 0	数値データ 3	
☒	3 E 2 1	EXP	
☒	3 E A 5	LOG	
	3 E F A	へ(べき乗)処理	
☒	3 F 5 1	COS	
☒	3 F 5 7	SIN	
☒	3 F 9 2	SQR	
☒	3 F D 3	TAN	
	3 F E 5 └ 3 F F F	} 不使用	

付-3 ワークエリア一覧表

アドレス	内 容
FA00 FA01	} DW 0FB2H 割り込み(未使用)
FA02 FA03	} DW 0EB5H キー割り込み③
FA04 FA05	} DW 0F4AH RS-232C 用割り込み
FA06 FA07	} DW 0F74H 2 msec タイマ割り込み
FA08 FA09	} DW 0F9FH CMT READ 割り込み
FA0A FA0B	} DW 0FB2H 割り込み(未使用)
FA0C FA0D	} DW 0FB2H 割り込み(未使用)
FA0E FA0F	} DW 0EA8H キー割り込み①
FA10 FA11	} DW 0EA8H キー割り込み①
FA12 FA13	} DW 0FB7H CMT エラー割り込み
FA14 FA15	} DW 0EB0H キー割り込み②
FA16 FA17	} DW 0F40H ゲーム用キー割り込み
FA18	STOP. ESC キー フラグ 03H=STOP 1BH=ESC 00H=それ以外
FA19	CMT 割り込みフラグ bit1=1 ならデータ受けとった, bit4=1 ならエラーが発生
FA1A	・入力先I/O機器指定 00H: キーボード 02H: RS-232C 80H: CMT ・PLAY においてはチャンネルを示す。 03H: チャンネル A 04H: チャンネル B 05H: チャンネル C
FA1B FA1C	} DW FB8FH ラウンドロビン バッファ制御用MAPの先頭アドレスを示す。
FA1D	CMT READ 割り込み用ワーク入力したデータが入る。

FA 1 E	CLOAD '*' の点減用フラグ前の状態が入る。																											
FA 1 F	CMT ボーレート指定 00H=600 ボー FFH=1200 ボー																											
FA 2 0	DB 10H CONSOLE の最大行数																											
FA 2 1	DB C3H																											
FA 2 2 FA 2 3	} LINE で使用																											
FA 2 4		DB C3H																										
FA 2 5 FA 2 6	} LINE で使用																											
FA 2 7		前にシステムポート B0H へ出力したデータ <table><tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td></td><td>/</td><td>/</td><td>/</td><td>/</td><td>MOTOR ON/OFF</td><td>VRAMアドレス 切換え</td><td></td><td>TIMER ON/OFF</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>0... OFF 1... ON</td><td>0 0...C 0 0 0 H 0 1...E 0 0 0 H 1 0...8 0 0 0 H 1 1...A 0 0 0 H</td><td></td><td>0... ON 1... OFF</td></tr></table>	bit	7	6	5	4	3	2	1	0		/	/	/	/	MOTOR ON/OFF	VRAMアドレス 切換え		TIMER ON/OFF						0... OFF 1... ON	0 0...C 0 0 0 H 0 1...E 0 0 0 H 1 0...8 0 0 0 H 1 1...A 0 0 0 H	
bit	7	6	5	4	3	2	1	0																				
	/	/	/	/	MOTOR ON/OFF	VRAMアドレス 切換え		TIMER ON/OFF																				
					0... OFF 1... ON	0 0...C 0 0 0 H 0 1...E 0 0 0 H 1 0...8 0 0 0 H 1 1...A 0 0 0 H		0... ON 1... OFF																				
FA 2 8) FA 2 B	} TIME 用データ																											
FA 2 C		プリンタ印字フラグ 00H:前にプリンタで印字を1回も行っていない 01H:印字を行なった																										
FA 2 D	CONSOLE 第4パラメータ(キーのクリック音発生)用フラグ 00H : ON 00H以外: OFF																											
FA 2 E	カーソル点減用フラグ																											
FA 2 F	カーソル点減状態フラグ 00H でノーマルな状態 FFH で反転の状態にある																											
FA 3 0	グラフィックキー用ワーク																											
FA 3 1	グラフィックキー用フラグ																											
FA 3 2	ファクションキーカウンタ																											
FA 3 3) FA 4 6	} ファンクションキー初期データ(第4章参照)																											

FA 3 3	}	COLOR	
FA 3 4			
FA 3 5	}	CLOAD	
FA 3 6			
FA 3 7	}	GOTO	
FA 3 8			
FA 3 9	}	LIST	
FA 3 A			
FA 3 B	}	RUN	CR
FA 3 C			
FA 3 D	}	SCREEN	
FA 3 E			
FA 3 F	}	CSAVE	
FA 4 0			
FA 4 1	}	PRINT	
FA 4 2			
FA 4 3	}	PLAY	
FA 4 4			
FA 4 5	}	CONT	CR
FA 4 6			
=20H(スペースコード)			
FA 4 7	}	乱数エリア	
FA 4 B			
FA 4 C	}	乱数初期値	0.499766207
FA 5 0			
FA 5 1	}	乱数エリア 2	前回の乱数が入っている。
FA 5 5			
FA 5 7	プリンタ・ヘッド位置 LPOS 用		
FA 5 8	出力先 I/O 機器指定 00H : CRT 01H : プリンタ 02H : RS-232C 80H : CMT		
FA 5 9	プリント命令のカンマ処理用定数		
FA 5 B	}	スタック・ポインタ初期アドレス	
FA 5 C			
FA 5 D	}	BASIC 実行中の行番号	
FA 5 E			

FA 5 F FA 6 0	} BASIC プログラム開始アドレス
FA 6 1) FB 3 C	} BASIC 命令 ジャンプテーブル

アドレス	命令名	中間コード	処理アドレス
FA 6 1, 6 2	END	8 0	3 5 3 5
FA 6 3, 6 4	FOR	8 1	0 6 7 E
FA 6 5, 6 6	NEXT	8 2	3 5 F 7
FA 6 7, 6 8	DATA	8 3	0 7 E 0
FA 6 9, 6 A	INPUT	8 4	0 9 A B
FA 6 B, 6 C	DIM	8 5	3 3 0 2
FA 6 D, 6 E	READ	8 6	0 A 0 9
FA 6 F, 7 0	LET	8 7	0 7 F 5
FA 7 1, 7 2	GOTO	8 8	0 7 A 0
FA 7 3, 7 4	RUN	8 9	0 7 8 1
FA 7 5, 7 6	IF	8 A	0 8 6 1
FA 7 7, 7 8	RESTORE	8 B	3 5 1 9
FA 7 9, 7 A	GOSUB	8 C	0 7 8 F
FA 7 B, 7 C	RETURN	8 D	0 7 B C
FA 7 D, 7 E	REM	8 E	0 7 E 2
FA 7 F, 8 0	STOP	8 F	3 5 3 3
FA 8 1, 8 2	OUT	9 0	0 D D 6
FA 8 3, 8 4	ON	9 1	0 8 4 4
FA 8 5, 8 6	LPRINT	9 2	0 8 7 A
FA 8 7, 8 8	DEF	9 3	0 D 3 A
FA 8 9, 8 A	POKE	9 4	0 D F A
FA 8 B, 8 C	PRINT	9 5	0 8 7 E
FA 8 D, 8 E	CONT	9 6	3 5 6 B
FA 8 F, 9 0	LIST	9 7	0 5 D B
FA 9 1, 9 2	LLIST	9 8	0 5 D 6
FA 9 3, 9 4	CLEAR	9 9	3 5 A 9
FA 9 5, 9 6	COLOR	9 A	1 D 9 B
FA 9 7, 9 8	PSET	9 B	2 D 3 C
FA 9 9, 9 A	PRESET	9 C	2 D 3 7
FA 9 B, 9 C	LINE	9 D	2 D C 7
FA 9 D, 9 E	PAINT	9 E	2 E D C
FA 9 F, A 0	SCREEN	9 F	1 E 0 4
FA A 1, A 2	CLS	A 0	1 D F 8
FA A 3, A 4	LOCATE	A 1	1 C D 2
FA A 5, A 6	CONSOLE	A 2	1 C F 6

FAA7, A8	CLOAD	A3	2496
FAA9, AA	CSAVE	A4	247E
FAAB, AC	EXEC	A5	261D
FAAD, AE	SOUND	A6	1E9B
FAAF, B0	PLAY	A7	1EB3
FAB1, B2	KEY	A8	2353
FAB3, B4	LCOPY	A9	22A6
FAB5, B6	NEW	AA	34CD
FAB7, B8	拡張用 No1	AB	
FAB9, BA	" No2	AC	
FABB, BC	" No3	AD	
FABD, BE	" No4	AE	
FABF, C0	" No5	AF	
FAC1, C2	" No6	B0	
FAC3, C4	" No7	B1	
FAC5, C6	" No8	B2	
FAC7, C8	" No9	B3	
FAC9, CA	" No10	B4	
FACB, CC	" No11	B5	
FACD, CE	" No12	B6	
FACF, D0	" No13	B7	
FAD1, D2	" No14	B8	
FAD3, D4	" No15	B9	
FAD5, D6	" No16	BA	
FAD7, D8	" No17	BB	
FAD9, DA	" No18	BC	
FADB, DC	" No19	BD	
FADD, DE	" No20	BE	
FADF, E0	" No21	BF	
FAE1, E2	" No22	C0	
FAE3, E4	" No23	C1	
FAE5, E6	SGN	D4	3898
FAE7, E8	INT	D5	39E7
FAE9, EA	ABS	D6	38B9
FAEB, EC	USR	D7	0755
FAED, EE	FRE	D8	32DE
FAEF, F0	INP	D9	0DCC
FAF1, F2	LPOS	DA	0D22
FAF3, F4	POS	DB	0D27
FAF5, F6	SQR	DC	3F92
FAF7, F8	RND	DD	3BA3
FAF9, FA	LOG	DE	3EA5

		FAFB,FC	EXP	DF	3E21
		FAFD,FE	COS	E0	3F51
		FAFF,00	SIN	E1	3F57
		FB01,02	TAN	E2	3FD3
		FB03,04	PEEK	E3	0DF3
		FB05,06	LEN	E4	3229
		FB07,08	HEX\$	E5	03EA
		FB09,0A	STR\$	E6	305B
		FB0B,0C	VAL	E7	32BA
		FB0D,0E	ASC	E8	3238
		FB0F,10	CHR\$	E9	3249
		FB11,12	LEFT\$	EA	3257
		FB13,14	RIGHT\$	EB	3286
		FB15,16	MID\$	EC	328F
		FB17,18	POINT	ED	} No内部 では使用し ていない。
		FB19,1A	CSRLIN	EE	
		FB1B,1C	STICK	EF	
		FB1D,1E	STRIG	F0	
		FB1F,20	TIME	F1	
		FB21,22	関数拡張用No1	F2	
		FB23,24	" No2	F3	
		FB25,26	" No3	F4	
		FB27,28	" No4	F5	
		FB29,2A	" No5	F6	
		FB2B,2C	" No6	F7	
		FB2D,2E	" No7	F8	
		FB2F,30	" No8	F9	
		FB31,32	" No9	FA	
		FB33,34	" No10	FB	
		FB35,36	" No11	FC	
		FB37,38	" No12	FD	
		FB39,3A	" No13	FE	
		FB3B,3C	" No14	FF	

FB3D	}	ファンクションキー1 データ (第4章参照)
FB44		
FB45	}	ファンクションキー2 データ
FB4C		
FB4D	}	ファンクションキー3 データ
FB54		

FB 5 5 ↓ FB 5 C	} ファンクションキー4 データ
FB 5 D ↓ FB 6 4	} ファンクションキー5 データ
FB 6 5 ↓ FB 6 C	} ファンクションキー6 データ
FB 6 D ↓ FB 7 4	} ファンクションキー7 データ
FB 7 5 ↓ FB 7 C	} ファンクションキー8 データ
FB 7 D ↓ FB 8 4	} ファンクションキー9 データ
FB 8 5 ↓ FB 8 C	} ファンクションキー10 データ
FB 8 D FB 8 E	} ファンクションキー・アドレスポインタ
FB 8 F ↓ FB 9 4	} キー入力用バッファ制御マップ
FB 8 F FB 9 0 FB 9 1 FB 9 2 FB 9 3 FB 9 4	<div> <div> バッファ内データ先頭の相対位置 バッファ内データ終了の相対位置 フラグ バッファの最大容量 (3FH, 63 バイト) } バッファの開始アドレス </div> <div> (*) </div> </div>
FB 9 5 ↓ FB 9 A	} RS-232C 用バッファ制御マップ (内容は(*)と同じ)
FB 9 B ↓ FB A 0	} ラウンドロビンバッファ制御マップ 拡張用 (内容は(*)と同じ)

FBA1 } FBA6	} PLAY チャンネル A 用バッファ制御マップ (内容は(*)と同じ)
FBA7 } FBAC	} PLAY チャンネル B 用バッファ制御マップ (内容は(*)と同じ)
FBAD } FBB2	} PLAY チャンネル C 用バッファ制御マップ (内容は(*)と同じ)
FBB3 } FBB8	} バッファ・フラグ処理用ワーク(詳細は不明)
FBB9 } FBF8	} キー入力 用バッファ (63バイト)
FBF9 } FC38	} RS-232C 用バッファ (63バイト)
FC39 } FC78	} PLAY チャンネル A 用バッファ (63バイト)
FC79 } FCB8	} PLAY チャンネル B 用バッファ (63バイト)
FCB9 } FCF8	} PLAY チャンネル C 用バッファ (63バイト)
FCF9 } FD1A	} PLAY 関係ワーク(詳細は不明)
FD1B	タイマ割り込み内 PLAY 用フラグ 00H で終了
FD1C } FD8B	} PLAY 各チャンネル用ワークエリア (71H バイト)
FD8C	最大ページ数 01H ~ 04H
FD8D FD8E	} BASIC エリアの上限

FD8F	(SCREEN 第2パラ)-1 アクティブ・ページ
FD90	(SCREEN 第3パラ)-1 表示ページ
FD91 ↓ FDC7	} アクティブページ・データ (**) <hr/>
FD91	V-RAM(アトリビュートを含む)の開始アドレスの上位(例 80H)
FD92	(SCREEN 第1パラ)-1 モード
FD93	COLOR 第1パラ
FD94	COLOR 第2パラ
FD95	COLOR 第3パラ
FD96 ↓ FD98 ↓ FD99 ↓ FD9B ↓ FD9C ↓ FD9E ↓ FD9F ↓ FDA1	} 現在のモードでの COLOR データ } モード1用 COLOR データ } モード2用 COLOR データ } モード3用 COLOR データ } モード4用 COLOR データ
FDA2	CONSOLE 開始行+1 (1~16)
FDA3	CONSOLE 最終行+1 (1~16)
FDA4	CONSOLE 開始行+1 (1~16)
FDA5	CONSOLE 最終行+1-(FDA6) (1~16)
FDA6	CONSOLE 第3パラ(ファンクションキー表示) 01H: ON 00H: OFF
FDA7	ファンクションキー表示状態(SHIFT による) 01H: f1~f5 03H: f6~f10
FDA8	カーソル Y 座標+1
FDA9	カーソル X 座標+1
FDA A ADAB	} V-RAM 上でのカーソルアドレス
FDAC	カーソル移動可能な X 座標+1の最大値 モード3のとき 10H モード1, 2, 4の時 20H
FDAD	NULL コード 20H
FDAE FDAF	} グラフィック命令での X 座標

FDB 0	} グラフィック命令での Y 座標
FDB 1	
FDB 2	
	COLOR データ
	(例)03H: モード1, 20H: モード2, C0H: モード3, 80H: モード4
FDB 3	} CRT アドレス
FDB 4	
FDB 5	} 不明 (DW 1010H)
FDB 6	
FDB 7	} 0~15行までの行フラグ
↓	
FDC 6	} スクリーンエディット用
FDC 7	
	不明
FDC 8	} ページ1用データエリア
↓	
FDFE	
	(FD91H~FDC7H までのデータが、コピーされる。)
FDF F	} ページ2用データエリア
↓	
FE 3 5	
FE 3 6	} ページ3用データエリア
↓	
FE 6 C	
FE 6 D	} ページ4用データエリア
↓	
FE A 3	
FE A 4	} コマンド入力時のカーソル相対位置
FE A 5	
FE A 6	コマンド入力時の X 座標最大値+1
FE A 7	不明
FE A 8	INS フラグ FFH: INS 中 00H: INS 中ではない。
FE A 9	LF の時のフラグ
FE A A	STOP した時の (SCREEN 第2パラメータ)-1
FE A B	STOP した時の (SCREEN 第3パラメータ)-1
FE A C	アトリビュートの DATA
FE A D	} グラフィック X 座標のコピー
FE A E	
FE A F	} グラフィック Y 座標のコピー
FE B 0	

FEB 1 FEB 2	} グラフィック関係ワーク
FEB 3 FEB 4	} LINE 用ワーク
FEB 5 FEC 5	} PAINT 用ワーク
FEC 6	不明
FEC 7	ファンクションキー表示で表示できる文字数
FEC 8	ファンクション表示モード
FEC 9	ファンクションのカウンタ
FEC A	SPACE, カーソル, STOP, SHIFT 用キーフラグ <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">bit 7</div> <div style="text-align: center;">6</div> <div style="text-align: center;">5</div> <div style="text-align: center;">4</div> <div style="text-align: center;">3</div> <div style="text-align: center;">2</div> <div style="text-align: center;">1</div> <div style="text-align: center;">0</div> </div> <div style="display: flex; justify-content: space-around; align-items: center; margin-top: 5px;"> <div style="border: 1px solid black; padding: 2px 5px;">スペース</div> <div style="border: 1px solid black; padding: 2px 5px;">00H</div> <div style="border: 1px solid black; padding: 2px 5px;">←</div> <div style="border: 1px solid black; padding: 2px 5px;">→</div> <div style="border: 1px solid black; padding: 2px 5px;">↓</div> <div style="border: 1px solid black; padding: 2px 5px;">↑</div> <div style="border: 1px solid black; padding: 2px 5px;">STOP</div> <div style="border: 1px solid black; padding: 2px 5px;">SHIFT</div> </div> <p>押されたキーのビットが1になる。</p>
FEC B FED 0	} CMT ファイルネーム バッファ 1 (キー入力より)
FED 1 FED 6	} CMT ファイルネーム バッファ 2 (CMT より)
FED 7	CMT マークチェックフラグ
FED 8	LOAD, VERIFY フラグ 00H : LOAD エラーすると NEW がかかる。 FFH : VERIFY
FED 9	キーバッファ用 00H 定数
FED A FF 2 1	} キーバッファ 72文字
FF 2 2 FF 2 3	} 不明
FF 2 4	DIM フラグ AFH : DIM 00H : 変数解析
FF 2 5	型フラグ 00H : 数値型 01H : 文字型

FF 2 6	ダブルクォーテーションまたは REM・DATA 出現フラグ
FF 2 7 FF 2 8	} BASIC エリア終アドレス
FF 2 9 FF 2 A	} BASIC エリア開始アドレス
FF 2 B FF 2 C	} 文字変数のポインタ
FF 2 D ↓ FF 3 8	} 不明
FF 3 9 FF 3 A FF 3 B FF 3 C	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> } 文字変数データ </div> <div> 文字数 グミー } ポインタ </div> </div>
FF 3 D FF 3 E	} 文字列フリーエリア先頭アドレス
FF 3 F FF 4 0	} 汎用ワーク
FF 4 1 ↓ FF 4 4	} 不明
FF 4 5 FF 4 6	} DATA 文 行番号
FF 4 7	FN フラグ
FF 4 8	047AH で使用 詳細は不明
FF 4 9	READ, INPUT フラグ 00H : INPUT 00H以外: READ
FF 4 A ↓ FF 4 D	} 不明
FF 4 E FF 4 F	} STOP, END の時の TEXT↑ アドレス } (LET で変数領域ポインタ用ワークとしても使用)
FF 5 0 FF 5 1	} 汎用ワーク

FF52 FF53	} STOP 時の行番号。
FF54 FF55	} 次に実行する TEXT のアドレス
FF56 FF57	} 変数領域の開始アドレス
FF58 FF59	} 配列領域の開始アドレス
FF5A FF5B	} フリーエリアの開始アドレス
FF5C FF5D	} DATA 文のポインタ
FF5E ↓ FF64	} FN で使用
FF65	ダミー
FF66 FF67 FF68 FF69 FF6A	<div> <div> } 仮数部 } 指数部 </div> <div> FAC①(文字列データの格納場所としても使用) </div> </div>
FF6B	補正用フラグ 80H: 正 7FH: 負 FFH: 整数
FF6C	ダミー
FF6D ↓ FF71	} FAC②
FF72 ↓ FF76 FF77 FF78 ↓ FF7B	<div> <div> } FAC③ </div> <div> } 10進変換で使用 </div> </div> <div> } FAC④ </div>
FF7C ↓ FF81	} 不明

FF82 } FF86	乱数エリア
FF87 } FF89	不明
FF8A } FFE3	フックエリア
FF8A } FF8C FF8D } FF8F FF90 } FF92 FF93 } FF95 FF96 } FF98 FF99 } FF9B FF9C } FF9E FF9F } FFA1 FFA2 } FFA4 FFA5 } FFA7 FFA8 } FFAA	0955H よりフック PRINT#, INPUT# 終了処理から 0404H よりフック エラー処理から 0415H よりフック エラー処理から 0442H よりフック TEXT EDIT から 0472H よりフック TEXT EDIT から 04C4H よりフック TEXT EDIT から 04CAH よりフック TEXT EDIT から 0578H よりフック 中間言語変換から 0665H よりフック 中間言語逆変換から 0716H よりフック 実行 MAIN ルーチンから 0781H よりフック RUN から

FFAB	}	0844H よりフック
FFAD		ON から
FFAE	}	087EH よりフック
FFB0		PRINT から
FFB1	}	08D2H よりフック
FFB3		PRINT から
FFB4	}	098BH よりフック
FFB6		INPUT, READ エラー処理から
FFB7	}	0995H よりフック
FFB9		INPUT, READ エラー処理から
FFBA	}	09B1H よりフック
FFBC		INPUT から
FFBD	}	0A46H よりフック
FFBF		READ, INPUT 共有ルーチンから
FFC0	}	0B81H よりフック
FFC2		式解析ルーチンから
FFC3	}	0C3BH よりフック
FFC5		式解析, 関数処理から
FFC6	}	2DC7H よりフック
FFC8		LINE から
FFC9	}	34E0H よりフック
FFCB		NEW から
FFCC	}	34CEH よりフック
FFCE		NEW から
FFCF	}	26C7H よりフック
FFD1		1 文字出力ルーチンから
FFD2	}	22A6H よりフック
FFD4		LCOPY から

FFD5	}	1951H よりフック カラー処理関係から
FFD7		
FFD8	}	0F31H よりフック キー割り込みから
FFDA		
FFDB	}	RST18H で使用
FFDD		
FFDE	}	現在 未使用
FEE0		
FFE1	}	RST38Hで使用
FFE3		
FFE4	}	未使用
FFFF		

付-4 中間言語と処理ルーチン対応表

キーワード	中間言語コード	処理ルーチン	HEX\$	E 5	0 3 E A
ABS	D 6	3 8 B 9	IF	8 A	0 8 6 1
AND	C F	0 C 9 A	INKEY\$	C 6	2 7 7 1
ASC	E 8	3 2 3 8	INP	D 9	0 D C C
CHR\$	E 9	3 2 4 9	INPUT	8 4	0 9 A B
CLEAR	9 9	3 5 A 9	INT	D 5	3 9 E 7
CLOAD	A 3	2 4 9 6	KEY	A 8	2 3 5 3
CLS	A 0	1 D F 8	LCOPY	A 9	2 2 A 6
COLOR	9 A	1 D 9 B	LEFT\$	E A	3 2 5 7
CONSOLE	A 2	1 C F 6	LEN	E 4	3 2 2 9
CONT	9 6	3 5 6 B	LET	8 7	0 7 F 5
COS	E 0	3 F 5 1	LINE	9 D	2 D C 7
CSAVE	A 4	2 4 7 E	LIST	9 7	0 5 D B
CSRLIN	E E	0 D 3 0	LLIST	9 8	0 5 D 6
DATA	8 3	0 7 E 0	LOCATE	A 1	1 C D 2
DEF	9 3	0 D 3 A	LOG	D E	3 E A 5
DIM	8 5	3 3 0 2	LPOS	D A	0 D 2 2
END	8 0	3 5 3 5	LPRINT	9 2	0 8 7 A
EXEC	A 5	2 6 1 D	MID\$	E C	3 2 8 F
EXP	D F	3 E 2 1	NEW	A A	3 4 C D
FN	C 4	0 D 6 1	NEXT	8 2	3 5 F 7
FOR	8 1	0 6 7 E	NOT	C 8	0 C F 9
FRE	D 8	3 2 D E	ON	9 1	0 8 4 4
GOSUB	8 C	0 7 8 F	OR	D 0	0 C 9 9
GOTO	8 8	0 7 A 0	OUT	9 0	0 D D 6

PAINT	9 E	2 E D C
PEEK	E 3	0 D F 3
PLAY	A 7	1 E B 3
POINT	E D	2 D 5 5
POKE	9 4	0 D F A
POS	D B	0 D 2 7
PRESET	9 C	2 D 3 7
PRINT	9 5	0 8 7 E
PSET	9 B	2 D 3 C
READ	8 6	0 A 0 9
REM	8 E	0 7 E 2
RESTORE	8 B	3 5 1 9
RETURN	8 D	0 7 B C
RIGHTS	E B	3 2 8 6
RND	D D	3 B A 3
RUN	8 9	0 7 8 1
SCREEN	9 F	1 E 0 4
SGN	D 4	3 8 9 8
SIN	E 1	3 F 5 7
SOUND	A 6	1 E 9 B
SPC	C 5	0 9 2 6
SQR	D C	3 F 9 2
STICK	E F	2 2 3 6
STEP	C 9	—
STOP	8 F	3 5 3 3
STR\$	E 6	3 0 5 B

STRIG	F 0	2 2 5 6
TAB	C 2	0 9 2 6
TAN	E 2	3 F D 3
THEN	C 7	—
TIME	F 1	1 E 8 3
TO	C 3	—
USR	D 7	0 7 5 5
VAL	E 7	3 2 B A
+	C A	3 6 7 E
—	C B	3 6 8 3
*	C C	3 7 B 4
/	C D	3 8 0 3
^	C E	3 E F A
>	D 1	—
=	D 2	—
<	D 3	—

付-5 キャラクタ・コード表

		上位4ビット															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
下 位 4 ビ ツ ト	0		π		0	Ⓐ	P		p	♠			—	タ	ミ	た	み
	1	月		!	1	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む
	2	火		"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め
	3	水		#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も
	4	木		\$	4	D	T	d	t	○	え	,	エ	ト	ヤ	と	や
	5	金		%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ
	6	土		&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
	7	日		'	7	G	W	g	w	あ	き	ア	キ	ヌ	ラ	ぬ	ら
	8	年		(8	H	X	h	x	い	く	イ	ク	ネ	リ	ね	り
	9	円)	9	I	Y	i	y	う	け	ウ	ケ	ノ	ル	の	る
	A	時		*	:	J	Z	j	z	え	こ	エ	コ	ハ	レ	は	れ
	B	分		+	;	K	(k		お	さ	オ	サ	ヒ	ロ	ひ	ろ
	C	秒	×	,	<	L	Y	l		や	し	ヤ	シ	フ	ワ	ふ	わ
	D	百	大	—	=	M)	m		ゆ	す	ユ	ス	ヘ	ン	へ	ん
	E	千	中	.	>	N	ハ	n	～	よ	せ	ヨ	セ	ホ		ほ	
	F	万	小	/	?	O	—	o		つ	そ	ッ	ソ	マ	。	ま	

付-6 キーボード配列表(1)

	CN7 X0	X1	X2	X3	X4	X5	X6	X7
CN6 Y0		CTRL	SHIFT	GRAPH				
Y1	1 ぬ	Q た	A ち	Z つ	K の	I に	8 yb yb	< , ね
Y2	2 ふ	W て	S と	X さ	L り	Q ろ	9 よ よ	> . る
Y3	3 あ	E い	D し	C そ	+ れ	P せ	F1	? / め
Y4	4 う	R す	F は	V ひ	: * け	@ "	F2	- ろ
Y5	5 え	T か	G き	B こ] む	[ろ	F3	SPACE
Y6	6 お	Y ん	H く	N み	= ほ	^ へ	F4	0 わ
Y7	7 や	U な	J ま	M も		Y ー	F5	
Y8	RETURN	STOP	↑	↓	→	←	TAB	ESC
Y9	かな	INS	DEL	()	HOME CLR			

付-6 キーボード配列表(2)

	CN7 X0	X1	X2	X3	X4	X5	X6	X7
CN6 Y0								
Y1	日						百	小
Y2	月		秒	×	中		千	大
Y3	火	厂	卜	厶	♣	π		♠
Y4	水	丁	十	丄	♥			♦
Y5	木	ㄣ	乚	ㄣ	●	○		
Y6	金	年	時		一			万
Y7	土			分		円		
Y8								
Y9								

付-7 エラーメッセージ一覧表

エラー コード	エラー	エラー メッセージ	意 味
0	N F	Next without For Error	NEXT が多すぎる。
2	S N	Syntax Error	文法がまちがっている。
4	R G	Return without Gosub Error	RETURN だけがある。
6	O D	Out of Data Error	DATA がありません。
8	F C	Function Call Error	規定範囲外の数を使った。
10	O V	Over flow Error	計算結果が大き過ぎる。または小さ過ぎる。
12	O M	Out of Memory Error	メモリ容量が足りません。
14	U L	Undefined Line Error	指定した行番号がありません。
16	B S	Bad Subscript Error	配列変数の添字が規定の範囲を超えている。
18	D D	Duplicate Definition Error	同じ配列を2度定義した。
20	/ 0	Division by Zero Error	0 で割算を行なった。
22	I D	Illegal Direct Error	INPUT, DEFFN をダイレクトモードで行なった。
24	T M	Type Mismatch Error	式の左右の型が一致していない。
26	O S	Out of String Space Error	文字列変数のエリアが足りない。
28	L S	Long String Error	文字列の長さが255文字を超えた。
30	S T	String too Complex Error	文字式が複雑すぎる。
32	C N	Continue Error	CONT はできません。
34	U F	Undefined Function Error	DEFFN で定義されていません。
36	T R	Tape Read Error	テープの読みこみが正しくない。
38	M O	Missing Operand Error	パラメータの指定が不完全。
40	F D	File Data Error	ファイルのデータ形式がまちがっている。

付-8 タイニー・モニタ

本書で解説されている機械語プログラムのキーインやメモリダンプを行なうためのツールとして "TINY MONITOR" プログラムを BASIC で作成しましたので、ここにそのリストと使い方を紹介します。

〈プログラム・リスト〉

```

10 REM TINY MONITOR
20 CLS:PRINT " <<<TINY MONITOR >>>"
30 PRINT "S — SET MEMORY"
40 PRINT "D — DUMP MEMORY"
50 INPUT A$
60 IF LEFT$(A$,1)="s" OR LEFT$(A$,1)="S" THEN 1
  00
70 IF LEFT$(A$,1)="d" OR LEFT$(A$,1)="D" THEN 3
  00
80 PRINT :GOTO 50
100 REM set memory
110 A$=RIGHT$(A$,LEN(A$)-1)
120 AD=VAL("&h"+A$)
125 IF AD<0 THEN AD=2^16+AD
130 FOR I=AD TO 65535 STEP4:GOSUB 400:A1$="":FOR
  J=0 TO 3:A=PEEK(I+J)
140 GOSUB 500:PRINT "-";
150 A$=INKEY$:IF A$=""GOTO 150
160 IF A$=" "THEN PRINT " ";:GOTO 290
170 IF VAL("&h"+A$)=0 AND A$<>"0"THEN PRINT :GOT
  0 50
180 A1$=A$:PRINT A$;
190 A$=INKEY$:IF A$=""GOTO 190
200 IF A$=" "THEN PRINT " ";:GOTO 290
210 IF VAL("&h"+A$)=0 AND A$<>"0"THEN PRINT :GOT
  0 50
220 A1$=A1$+A$:PRINT A$; " ";A=VAL("&h"+A1$):POK
  E I+J,A:GOTO 290
290 NEXT J:PRINT:NEXT I:PRINT :GOTO 50
300 REM DUMP MEMORY
310 A$=RIGHT$(A$,LEN(A$)-1)
320 AD=VAL("&h"+A$)
325 IF AD<0 THEN AD=2^16+AD
330 FOR I=AD TO 65535 STEP8:GOSUB 400:A1$="":FOR
  J=0 TO 7:A=PEEK(I+J)
340 GOSUB 500:PRINT " ";
350 A$=INKEY$:IF A$<>" "THEN PRINT :GOTO 50
360 NEXT J:PRINT :NEXT I:PRINT :GOTO 50
400 A=INT(I/256):GOSUB 500:A=I-(INT(I/256)*256):
  GOSUB 500
410 PRINT " ";:RETURN
500 A1=INT(A/16):A2=A-(A1*16)
510 IF A1>=10 THEN A1=A1+7
520 A1=A1+&H30
530 IF A2>=10 THEN A2=A2+7
540 A2=A2+&H30
550 PRINT CHR$(A1)+CHR$(A2):RETURN

```

《使い方》

プログラムをキーインして、RUN させると、次のように表示され、入力待ちになります。

```

<TINY MONITOR>
S ..... SET MEMORY
D ..... DUMP MEMORY
?
```

機械語を書き込む場合は S を、メモリーダンプを行なうには D を選びますが、S や D の後に続けて、アドレスを16進で指定する必要があります。

```
ex) S C000
    d BF00
```

書き込みの際、アドレスに既書き込まれている値が表示されますが、これを変更する必要がない場合、スペースバーを押すと、次のアドレスの入力に移ります。

S または D のモードから抜けだすには、16進キー(0 ~ F)以外のキーを押します。

付-9 12平均率音階表

	1	2	3	4	5	6	7	DATA
C	32.70	65.41	130.81	261.63	523.25	1046.50	2093.01	32.70
C #	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	34.65
D	36.71	73.42	146.83	293.67	587.33	1174.66	2349.02	36.70
D #	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	38.89
E	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	41.20
F	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	43.65
F #	46.25	92.50	185.00	370.00	739.99	1474.98	2959.96	46.25
G	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	49.00
G #	51.91	103.83	207.65	415.31	830.61	1661.22	3322.44	51.91
A	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	55.00
A #	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	58.26
B	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	61.72

付-10 サウンドレジスター一覧表

レジスタ		ビット	B 7 MSB	B 6	B 5	B 4	B 3	B 2	B 1	B 0 LSB
R ₀	チャンネルA 音階	8bit Fine Tune A								
R ₁								4bit Coarse Tune A		
R ₂	チャンネルB 音階	8bit Fine Tune B								
R ₃								4bit Coarse Tune B		
R ₄	チャンネルC 音階	8bit Fine Tune C								
R ₅								4bit Coarse Tune C		
R ₆	ノイズ周波数					5bit Period Control				
R ₇	イネーブル	IN/OUT			ノイズ			トーン		
		I/O B	I/O A	C	B	A	C	B	A	
R ₈	チャンネルA 音量					M	L ₃	L ₂	L ₁	L ₀
R ₉	チャンネルB 音量					M	L ₃	L ₂	L ₁	L ₀
R ₁₀	チャンネルC 音量					M	L ₃	L ₂	L ₁	L ₀
R ₁₁	エンベロープ周期	8bit Fine Tune E								
R ₁₂		8bit Coarse Tune E								
R ₁₃	エンベロープ形状						CONT.	ATT.	ALT.	HOLD
R ₁₄	I/Oポート A データ・ストア	8bit パラレル I/O ポート A								
R ₁₅	I/Oポート B データ・ストア	8bit パラレル I/O ポート B								

索引

A

Acc	75
A/G	58
A/S	58
ASCIIコード	25・31・58
AUTO	146

B

BF命令	135
BUG	134

C

CG	62
CG ROM	62
CHRS	77
CLOAD PRINT	158
COLOR	132
CONSOLE	134
CPU	11・65
CRT	11・149
CRTC	132
CRTコントローラ	53
CSAVE	28
CSS	55・136
CTRL キー	77

D

DEレジスタ	145
DIM	41
DMA	65

E

EXEC	144
------------	-----

F

FAC	75・148
FD Error	111
FN	37
FOR	93
FRE関数	46

G

GOSUB	25
GOTO	25・158

H

HLレジスタ	145
--------------	-----

I

IC	53
IF	74
INKEY\$	37
INPUT	73
INT/ENT	57
INV	56
I/Oポート	74・97
I/Oマップ	18

K

KEY コマンド	70
KEY LIST	70・151

L

LCOPY	118・120
LINE	131
LIST	27
LOAD	22・104・108・110
LPRINT	166
LSI	85・88

M

MML	81
-----------	----

N

N-BASIC	25・129
N ₆₀ -BASIC	21・129
NEW	28
NEXT	93
NOT	37

O

OR	135
OUT	62

P

PC-6021	117
PC-8023	123・124
PEEK	143・167
PLAY	81
PLAYのバッファ	93
POINT	37
POKE	143・167
PRESET	132・171
PRINT	111・166
PROM	15

PSET	132-171
R	
RAM	11-15
RESET	28
ROM	11-15-69
ROM & RAM カートリッジ	15
RUN	35
S	
SAVE	22-104-108-110
SCREEN 関数	162
SOUND	88
SPC	37
STICK	74-144
STOP	147
STRIG	74
T	
TAB	37
TIME	37-155
TM Error	148
U	
USR	144-147
V	
VDG	11-53
VRAM	59-136

ア	
アトリビュート	55-59-136
アトリビュートエリア	55-59
アドレス	29-39-41
アドレスラッチ	97
アルファニューメリック	53
アルファニューメリック表示	53
アペンド	168
ジャンスト	159
イニシャライズ	28-35-70
インタフェイス	13
インタプリタ	11-35
ウィンドウ	71
エンドマーク	29-69
エンベロープジェネレータ	89

音楽用言語	81
カ	
拡張 BASIC	35
拡張 ROM エリア	14-165
カセットインタフェイス	11
ガベージコレクション	45
キー入力ステートメント	73
キーバッファ	76
キーフラグ	71
キーポインタ	72
キーボード	13
キーリビート	76
キーワード	31-69
機械語	70-97-110-143-144-150
キャラクタ・コード	38-77
キャラクタ・ジェネレータ	53-57
キャラクタ・フォント	62
キャラクタ・ROM	11
クロック	11
グラフィック	11-53
グラフィックアドレスマップ	60
グラフィックキャラクタ	38
グラフィック表示	53
コマンド・ステートメント	35
コントロールキー	77
コントロールコード	38-71-77
コンパチブル	11
コンペア命令	37
サ	
サウンド機能	81
サウンドレジスタ	97
サブ CPU	74
式解析ルーチン	37
識別コード	38
ジャンプテーブル	35-166
ジャンプ線	17
出力ポート	63
ジョイスティック	74
処理アドレス	37
処理ルーチン	35
ストリングディスクリプタ	43

セミグラフィック	53・58
タ	
ダイレクト・メモリ・アクセス	65
単純変数	39
単純変数領域	39
単精度	26・38・130
チャンネル	83・89
中間言語	31・108・164
中間言語処理ルーチン	35
データ・テープ	106
データ・ファイル	106・111
データ・フォーマット	106
テープフォーマット	112
テキストコンバータ	130
テキスト・セミグラフィック	59
ハ	
倍精度	26・27・130
配列変数領域	41
パラメータ	57
ファンクションキー	13・69・150
ファンクションキー・フラグ	71
フォーマット	104
浮動小数点	49
浮動小数点表記法	49
フリーエリア	46
プリンク機能	131
プリンタ・インタフェイス	13
プログラムエリア	14
ページ数	11
ヘッダ	106
ポインタ	22・43
ポーレート	103
マ	
マスク	11
メモリ	11
メモリダンブ	29
メモリ・マップ	21
文字型変数	40
文字配列	42
文字変数	43
文字列	72

文字列領域	43
モニタ・ROM	143
ヤ	
ユーザーエリア	22
ユーザープログラム	22
ラ	
リロケータブル	30
リンクポインタ	24・29
ルーチン	93
レジスタ	88
ワ	
ワークエリア	35・174

〈著者代表略歴〉

かずと れい
一 零

昭和30年8月1日 博多生まれ

昭和51年 九州産業大学電気工学科卒業

現在株式会社システムソフト福岡技術顧問

パーソナルコンピュータのハードウェア・ソフトウェア
の両方に精通し、特に機械語でのプログラム開発に特異
性を発揮、"頭の中"でアセンブル・逆アセンブルを行な
うため、さすらいのアセンブラーと呼ばれている。現在、
史上最大のパソコン・ウォーゲームを開発中。

本書の内容に関しまして、ご質問・ご意見がござい
ましたら下記の出版部まで書面にてお寄せくださいま
すようお願い致します。

PCファミリー・テクニカル・ノウハウ集

PC-6000シリーズ編

PC-Techknow 6000 Vol. 1

1982 ©システムソフト

検印廃止

共 著 一 零・樋口 理
八木 良一
監 修 システムソフト
発行者 塚本 慶一郎

発行所 株式会社 アスキー出版

〒150 東京都渋谷区神宮前5-2-2 瀬川ビル

振替 東京7-57496

☎03-407-4910(業務部) 03-407-4903(出版部)

1982年9月30日 第1版第1刷発行 Printed in Japan

2,500円

本書の一部あるいは全部について、株式会社アスキー出版から
文書による許諾を得ずに、いかなる方法においても無断で複写、
複製することは禁じられています。

ISBN4-87148-287-1 C3055 ¥2500E

監修
 SYSTEM SOFT
発行
 アスキー出版

定価2,500円

ISBN4-87148-287-1 C3055 ¥2500E